NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

DATA STRATEGIES TO SUPPORT AUTOMATED MULTI-SENSOR DATA FUSION IN A SERVICE ORIENTED ARCHITECTURE

by

Kurt J. Rothenhaus

June 2008

Dissertation Supervisor: James Bret Michael

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2008	**3. REPORT TYPE AND DATES COVERED** Dissertation	
4. TITLE AND SUBTITLE: Data Strategies to Support Automated Multi-Sensor Data Fusion in a Service Oriented Architecture			**5. FUNDING NUMBERS**
6. AUTHOR: CDR Kurt Rothenhaus			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

The quantity of data available to decision makers of various types is rapidly expanding beyond the pace of manual interpretation techniques (Hobbins, 1). Introducing a Service Oriented Architectures (SOA) based web service framework that exposes even more data without sufficient guidance will exacerbate the situation. Ontology's, data descriptions and discovery methods alone are not enough to create the end-to-end solutions promised by SOA technologies. Software architectural patterns in conjunction with broad data strategies are required to harness and employ vast quantities of content. This dissertation provides two software architectural patterns and an auto-fusion process that guide the development of a distributed, accountable and scalable SOA framework to support improved control and monitoring software. Although applicable to a wide range of software control system challenges, the dissertation will focus on a Maritime Domain Awareness (MDA) interoperability challenges. Using the U.S. Navy's MDA project as a case study, this dissertation will design, build and test a prototype automated data fusion framework employing the trickle-up and Command and Control Zone pattern that automates the discovery, pedigree assessment and ultimate fusion of dissimilar data types in a SOA web-service supported framework.

14. SUBJECT TERMS : Fusion, Multi-Sensor, Service Oriented Architectures, Software Engineering, Common Operations Picture, FORCEnet, Software Framework, Web Service Definition Language (WSDL), Universal Discovery Directory Index (UDDI), Simple Order Access Protocol (SOAP), Java, Maritime Domain Awareness (MDA), Business Process Execution Language for Web Service (BPEL4WS)	**15. NUMBER OF PAGES** 240
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18-298-102

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DATA STRATEGIES TO SUPPORT AUTOMATED MULTI-SENSOR DATA FUSION IN A SERVICE ORIENTED ARCHITECTURE

Kurt Joseph Rothenhaus
Commander, Program Executive Office C4I
Navy Program Office for Command and Control (PMW-150), San Diego, CA.
B.S., University of South Carolina, 1992
M.S., Naval Postgraduate School, 1999

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2008

Author: _____
Kurt Joseph Rothenhaus

Approved by:

_____	_____
James Bret Michael Professor of Computer Science Dissertation Supervisor Committee Chairman	Man-tak Shing Associate Professor of Computer Science
_____	_____
Ted Lewis Professor of Computer Science	John Osmundson Research Associate Professor of Information Science

_____
Dave Engel
Senior Scientist, Northrup Grumman

Approved by: _____
Peter J. Denning, Chairman, Department of Computer Science

Approved by: _____
Orrin D. Moses, Associate Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The quantity of data available to decision makers of various types is rapidly expanding beyond the pace of manual interpretation techniques (Hobbins, 1). Introducing a Service Oriented Architectures (SOA) based web service framework that exposes even more data without sufficient guidance will exacerbate the situation. Ontology's, data descriptions and discovery methods alone are not enough to create the end-to-end solutions promised by SOA technologies. Software architectural patterns in conjunction with broad data strategies are required to harness and employ vast quantities of content. This dissertation provides two software architectural patterns and an auto-fusion process that guide the development of a distributed, accountable and scalable SOA framework to support improved control and monitoring software. Although applicable to a wide range of software control system challenges, the dissertation will focus on a Maritime Domain Awareness (MDA) interoperability challenges. Using the U.S. Navy's MDA project as a case study, this dissertation will design, build and test a prototype automated data fusion framework employing the *trickle-up* and Command and Control *Zone* pattern that automates the discovery, pedigree assessment and ultimate fusion of dissimilar data types in a SOA web-service supported framework.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I wish to thank the Navy Program Office for Command and Control and the Navy Program Office for Intelligence Surveillance and Reconnaissance, John Shea (PMW-120), CAPT D.J. Legoff and Greg Settelmayer (PMW-150) for sponsorship of this research. Additionally, I wish to thank Mr. Steve Shell, Dr, Jeff Lansing, Dr. Frank White, Diana Akins, Dr. Otto Kessler and Dr. Marv Langston for their advice and assistance in understanding data fusion and software development. Furthermore, I wish to thank RADM Michael Mahon, USN, for his at-sea instruction on Command and Control doctrine.

I would like to thank Lieutenant Colonel Tom Cook for his thorough review of the research, his friendship and generosity. Furthermore I wish to thank the committee members for their guidance during the research. Special thanks to Professor Shing for providing critical instruction in software patterns and Unified Modeling Language. Furthermore I wish to thank Professor Ted Lewis for introducing me to Command and Control software development and his generous support and instruction, first as my computer science master's thesis advisor in 1999 and as a member of the dissertation committee.

I wish to thank Dr. Dave Engel for his generosity and wisdom in the conduct of this research and serving on the committee. It was an honor and a privilege to work with one of the leaders in the Command and Control community. Furthermore I wish to thank Professor James Bret Michael for his mentorship, instruction and brilliance. His dedicated efforts were the foundation for the study and his tireless efforts both as an educator and a researcher were critical during all phases of the research.

Finally I wish to thank my wife Momoko and daughters Mia and Nina for their patience, support and love.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The Department of Defense (DoD) Integrated Architecture Panel defines software architecture as "the structure of components, their relationships, and the principles and guidelines governing their design and evolution over time" (DoDAF, D-1). The DoD Net-Centric Data Strategy details goals to expand data management beyond standardization towards increasing data "visibility and accessibility" (Net-centric, 2003). A data strategy serves in large organizations such as the DoD, as guidance in the development of architectures to foster interoperability between systems that have independent requirements, sponsors, and funding streams. Data strategies and architectures support data visibility and accessibility, by guiding the development of interoperable software systems. Developing a suitable enterprise data strategy is challenging, as policy-makers balance the demands to standardize, while avoiding an overly limiting policy which would likely discourage innovation and improvement. A balanced data strategy should foster basic interoperability standards that make DoD data visible and accessible, by lowering the barrier of entry of authorized consumers to the content.

Systems of systems are normally comprised of multiple independent services to satisfy a desired behavior or need. Systems such as the Navy's AEGIS program integrated a number of unique systems to create complex systems of systems which included missiles, fire control, hull and mechanical among others. The high cost of integrating new capabilities is the primary motivation for moving towards a more loosely coupled binding between those systems. Web-enabled Service Oriented Architecture (SOA) can be viewed as a new phase in the evolution of software architectures, that being from standalone single-use embedded systems, to today's web-enabled systems (PRESSMAN, 12). SOA consists of discrete services, choreographed and linked to create added value by employing an overarching discovery and orchestration technology.

If data strategies provide overarching guidance in the follow-on development of architecture and design, then what would be an example of a data strategy concept of sufficient magnitude? For example, one of the critical requirements of the Internet

Protocol (IP) when the Defense Advanced Research Project Agency (DARPA) designed the infant Internet (known as ARPANET) was resilience to attack. Specifically, if a node or a link was destroyed in a nuclear attack, the system would continue to function and move data to its ultimate destination, albeit in a degraded mode of operation. The Internet resilience strategy requirement drove many other implicit and explicit design and architecture decisions about such things as unique addressing logic for each packet, router handling and Domain Name Services (DNS). These requirements manifested in a scalable and robust communications infrastructure for a broad range of information systems.

Examining a DoD data strategy for Joint Maritime Domain Awareness (MDA) requirements might include high-level goals to handle the increased quantities and variety of data (Steinberg, 2-2). One mechanism to handle the increased data quantity and variety includes architecting systems in such a manner as to facilitate auto-fusion. Fusion relates to the concept of taking two dissimilar data elements and combining them to create information (Hall, 1-1). For example, in a military context two system outputs such as one radar generated surface track, paired with an acoustic report can assist operators in categorizing a target. In a commercial example, a sales report and a weather report can be paired to reveal trends in sales as it relates to climate. Auto-fusion moves beyond traditional fusion, by automatically pairing data sources with fusion algorithms, which are optimized for the specific tactical environment.

Traditionally, data fusion was accomplished in a single system where developers had significant control of data definitions, timing, and the use of the system output (Bowman, 16-2). In today's Intelligence, Surveillance and Reconnaissance (ISR) system's attempting to fuse data from a set of heterogeneous systems is a challenge as each system may define its data objects differently, employ different communications protocol's and in many cases operates on different computing platforms. Combining data from various systems to increase situational awareness can involve complex and costly reengineering of the legacy system or comprising the ISR system of systems. A SOA based auto-fusion system must have a communication mechanism that provides for the mapping of one data set to another, a method for one service to discover the existence of

other related services and fusion engine's and an overarching mechanism to initiate and orchestrate the activities throughout the process.

Conducting multi-source fusion in a SOA has a number of challenges and benefits. Traditional ISR systems tend to have tightly integrated fusion logic with the presentation, persistence, and other processing logic residing in a single system. Although traditional ISR systems performed analysis relatively quickly and efficiently, the rigidity and tight-coupling of these systems limits their ability to address changing tactical situations or incorporate updated algorithms and new sources. In contrast, with SOA-based ISR systems, fusion algorithm logic can be reused and exchanged with a variety of data sources in a *plug and fight* framework. A framework in this case is the collection of data sources, data algorithms, along with binding, management, and persistence services tailored to provide situational awareness. A number of challenges exist to fuse data in a distributed system which is compounded in a SOA. These challenges are often referred to as *data alignment,* which includes common formatting, time propagation, misalignment compensation, and evidential conditioning (Bowman, 16-30). Common formatting refers to the machine-to-machine common language; time propagation relates to the criticality of time as a critical component of many fusion processes; coordinate conversion refers to the ability to employ various standards in data descriptions; misalignment compensation refers to the ability to adjust the process for known errors; and evidential conditioning refers to employing measures of probability or other likelihood in adjusting a process output.

To address some of the challenges mentioned above, this dissertation introduces two software architectural patterns and an auto-fusion process to influence elements of the Navy's Maritime Domain Awareness data strategy, with a focus on fostering auto-fusion in a SOA multi-source data framework. Software design patterns are proposed since the constructs are applicable to more than one instance and have applicability across the control and monitoring software domain. Software design patterns are high-level abstractions that provide the "why, where and how, not just what" for software design (Buschmann, 4). The hypothesis for the research present in this dissertation is that the two patterns and the auto-fusion process, applied in a SOA-based control-and-

monitoring software domain, enhance our ability to manage the significant quantities of data expected in a SOA environment. The research reported here is applicable to a wide spectrum of control and monitoring software challenges, such as industrial process monitoring or medical diagnostic systems. The patterns and process enhance the ability of systems and users to better manage remote sensor data and establish accountability and pedigree constructs for the control and monitoring software domain. Additionally, the research provides unique software patterns for the SOA distributed control and monitoring domain as well as a novel approach to data service management, description, and discovery in a SOA to improve situational awareness across a broad range of systems. The research builds on existing software design, maintenance, and management theory and practice by introducing new patterns for SOA base d systems, unique views on code reuse, and architectures that enable management of the evolution of software systems.

MDA is a good application domain in which to conduct SOA research. MDA requires combining various data sources with a wide range of pedigree and processing methods in a SOA framework. To address the heterogeneous nature of the data sources, various analysis algorithms and wide variety of data management methods, the research proposes two software design patterns; *trickle-up* and *zone*. The patterns are applied in an MDA command and control environment to explore the potential benefits of employing the patterns. The projected benefits of the *trickle-up* pattern are to better organize SOA services to enable dynamic discovery and improve software reuse. The *Zone* pattern potentially provides scalable methods to manage interface and partner dependencies. The *zone* and *trickle-up* pattern combined in a SOA enables an Auto-fusion process to discover and orchestrate data sources to improve user situational awareness in an MDA mission. A challenge of auto-fusion is its demand for seamless data exchange and the right processing logic at various levels of the architecture, which is discussed in greater detail in chapter four. The prototype auto-fusion framework developed in this dissertation will operate on a web-service platform; utilizing open standards based technologies such as orchestration and messaging. The resulting prototype examines methods to automate the discovery of new data sources, pedigree

assessment and ultimate fusion of dissimilar data types in service oriented architectures. In this dissertation, we start by determining the location and type of processing logic required for auto fusion. Next we develop a data model described as the *trickle-up* pattern that supports auto data fusion on a web-service platform to include data-exchange requirements between services, ontology definitions for discovery, and business-process statements to orchestrate fusion activities. We then provide a *zone* pattern for employing the trickle-up content. We demonstrate our approach via a prototype auto-fusion system on an enterprise web platform such as Sun Application server and simulate Maritime Domain Awareness Pilot's core and sensor services to validate the model.

A. TRICKLE-UP AND ZONE PATTERNS FOR COMMAND AND CONTROL IN A SERVICE ORIENTED ARCHITECTURE FRAMEWORK

Commander of the United States Pacific Fleet, Vice Admiral Robert F. Willard, defines C2 as the following; "Command is the doctrinal assignment of authority and Control is defined as guiding the operation" (Willard, 1). In his paper the *Art of Command and Control*, VADM Willard discusses the roles of commanders at the strategic and operational level of conflict to; maintain alignment, provide situational awareness, advance the plan, comply with procedure, counter the enemy and adjust apportionment (Willard, 2). Supporting his commander's roles, VADM Willard discussed C2 systems roles to provide a means to exchange relevant and timely information (Willard, 2). The exchange of timely and relevant information bridges two system domains; ISR and C2. In general ISR provides battlespace awareness and sensing content, and the C2 system provides commanders context and intentions. The crossroads of these systems is often referred to the Common Operational Picture (COP). The COP is where intelligence overlays the perceived location and intentions of the enemy, with the laydown of blue forces and the commanders plan. The COP is then replicated throughout the force as a means to communicate relevant and timely information, in an actionable format. This dissertation research focuses on the application of SOA technology on the

COP by providing software patterns and processes that address data fusion by means of the *trickle-up* pattern and the COP data management and synchronization by means of the zone pattern.

A number of concepts support the capability to autonomously fuse dissimilar data from differing sensors in a defense related enterprise framework. Firstly, service oriented architectures enable content from various sensors to reside as a discoverable discrete source. Secondly, using the Joint Directors of Laboratories (JDL) four-phase functional fusion model we can map established definitions and processes within the fusion domain to the command and control(C2)/ISR models described in this research (Llamas, 12). Thirdly, applying general concepts of knowledge management allows the proper assignment of data definitions and categories essential to architecting an enterprise approach to data fusion. Knowledge management's progressive levels such as data to information to knowledge can be adapted to C2 track-centric systems with a report to track-to-entity structure (extensible Tactical C4I Framework, 8) shown in Figure 1 below.

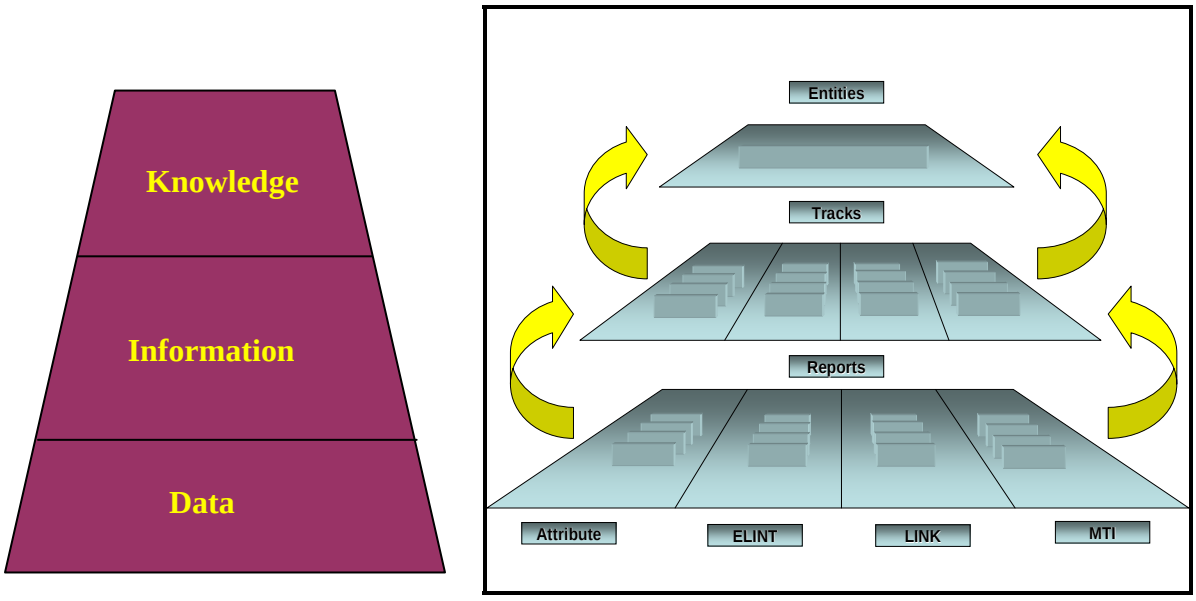

Figure 1. Knowledge Level-Trickle-up Pattern Mapping

Moving up from data to information we have data about the data which is metadata. Following the information level we start to apply the information to specific problems which graduates to knowledge which provides methods to apply information to solve a

problem. Reports are not always simple atomic data elements and an entity is a broad term to define relations between various types of information that may not be knowledge. For example at the first tier, data-level knowledge maps to reports. Reports tend to be atomic-level defense-related information such as a frequency at a certain bearing or a radar return as a specific bearing and range. These report-level objects are the building blocks for the next level of information, known as tracks. Tracks are objects usually associated with kinematics, such as the vectors (direction and velocity). The identity of the track object is often determined by meta-data from the track such as "its an aircraft since it has altitude" or "the track is a fighter aircraft because its going over 1000 mph" The next level of the information data model is knowledge and we map that to the Entity level in the *trickle-up* pattern. Knowledge is the ability to take data and information, analyze them, and in some cases predict outcomes or execute more complex analysis. Knowledge is a difficult concept to map to a data model and recognizing the varied nature of knowledge a broadly defined term such as "Entity" is used. An example of an entity-like object is the fusion of an acoustic object identified as a submarine's course, speed, and position with a piece of intelligence on the tactical preference of the submarine's commanding officer. This broad definition of an entity permits the essential flexibility in employment. Figure 2 is an example of ELINT (electrical signals intercepts) and acoustic-report objects fused into an anti-submarine warfare (ASW) track, which is then fused into a strategic ASW object when combined with strategic intelligence objects.

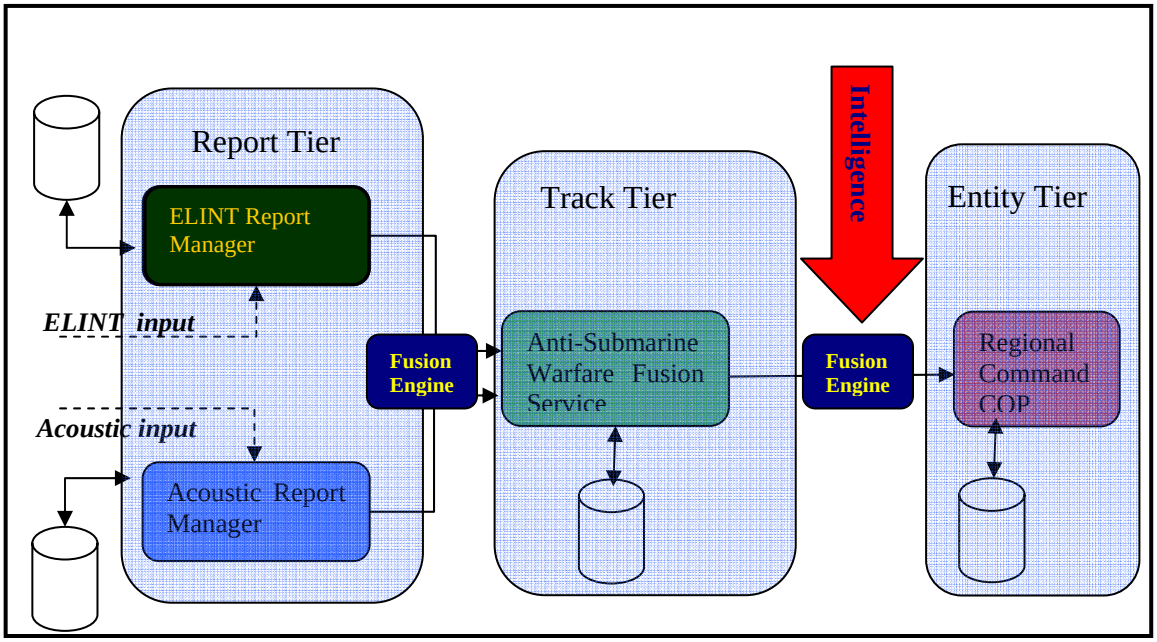

Figure 2. Trickle-up Pattern Example

The model above could exist in a number of systems, to include traditional vertically integrated systems that handle the data streams from sensor to human-computer interface (HCI) hard coding the fusion logic as required. A SOA-based model permits a more distributive architecture enabling:

- Open standard interface to each level of contents
- Loosely coupled services which can be configured during runtime in response to changing requirements
- Multiple "fusion engine" services to connect at runtime, allowing optimization of the fusion process by employing the best service components (e.g. algorithms) for the data involved
- Service reuse through employment of a service beyond initial planned use

Although some interesting analogues exist between the knowledge level and the *trickle-up* pattern discussed above, a one-to-one mapping for each level does not exist, though some overlap in definitions is present.

B. GOALS OF RESEARCH AND ADVANCES IN SOFTWARE ENGINEERING

The problem the dissertation is addressing concerns that in a SOA environment vast quantities of data will be exposed and present methods of data management are inadequate to manage it (Hobbins, 1). The hypothesis is that data ontologies, meta-data and discovery are not enough to address the volume of data expected to manifest itself in a SOA framework where significant data of various types are exposed and the two patterns and process proposed in the research can aid in managing the data. The research takes a holistic approach, starting with the two software patterns and auto-fusion process, then building prototypes, and finally providing input to the Maritime Domain Awareness data strategy to support an auto-fusion framework. The *trickle-up* pattern addresses the data source and fusion engine layer, the *zone* pattern addresses the relationships between the consumer of the *trickle-up* content and users, and the *auto-fusion* process provides a unifying method to dynamically manage the data.

Specifically the research involved developing:

- *Trickle-up* software pattern and data model to organize SOA services and promote dynamic discovery and binding.
- A *trickle-up* prototype to examine the location of processing logic in a web-enabled SOA architecture to support separating fusion engine from data source. The prototype system is built using Enterprise Java Beans and Java Servlet technology to include a XML based transport methods.
- A Command and Control *Zone* pattern to provide a scalable method to manage data, user, and consumer interfaces and dependencies as well as impacts on alternate Common Operations Picture topologies that result from the pattern.
- A demonstration using web-standard Business Process Execution Language for Web-services (BPEL4WS) to orchestrate the fusion process via a prototype system.

- An auto-fusion discovery service for fusion candidate selection. Propose an auto-fusion figure of merit to compare various service pairs and fusion algorithms.
- An examination of auto-fusion as an overarching data strategy to support improved Maritime Domain Awareness goals.

This research will provide guidance to the ISR community on the employment of a SOA to support improved situational awareness and decision making. Additionally, auto data fusion has commercial as well as military implications in the development of SOA. Systems developed today often involve a hub and spoke architecture where data from various sources are pulled into a central processing system for analysis. The research in this paper examines the use of a more distributed architecture where processing occurs at many different locations throughout the topology from various sources. The research applies to the general knowledge-management domain and provides an enterprise approach to information association and pattern matching enabling an automated mechanism to create a new data service comprised of two or more original sources. The research will examine potential implications on enterprise data management created from auto data fusion, and provide supporting artifacts for an MDA data strategy.

C. TECHNOLOGIES, RESEARCH STRATEGY AND METHODS

The goal of developing the prototype software is to validate the concept of auto-fusion in a SOA and provide a reference implementation for the two software patterns. The model employs a number of SOA web service technologies at various levels of maturity. Thomas Erl defines SOA as "baseline distributed architecture with no reference to implementation" (Erl, 1). He continues that coupled with web-service based technologies a real platform for "federation, agility and cross-platform harmony emerges" (Erl, 2). The basic principle of SOA involves two or more services exchanging data in a manner that supports reuse of the service. In many ways SOA is an abstraction or extension of object oriented programming with each service having a defined interface, a standardized communication method, and a method to discover various services. Sarukkai and Cohen discuss the position of SOA enabled web-services

in the Open Systems Interconnect (OSI) stack (Sarukkai, 3). Level six is the presentation level, which encompasses the protocol, data conversion, and transport policies. The communications methods web-services employ, such as simple order access protocol (SOAP), fit into the level six definitions. Figure 3 below shows the relative position in the OSI model, contrasted to Grahams proposed SOA "stack," explained later in this section. The web applications would reside on level seven or application level, and the session level or level five remains the transport protocols. Later in the section, an adaptation of the OSI model directed specifically at the various technologies comprising web-service SOA is discussed.

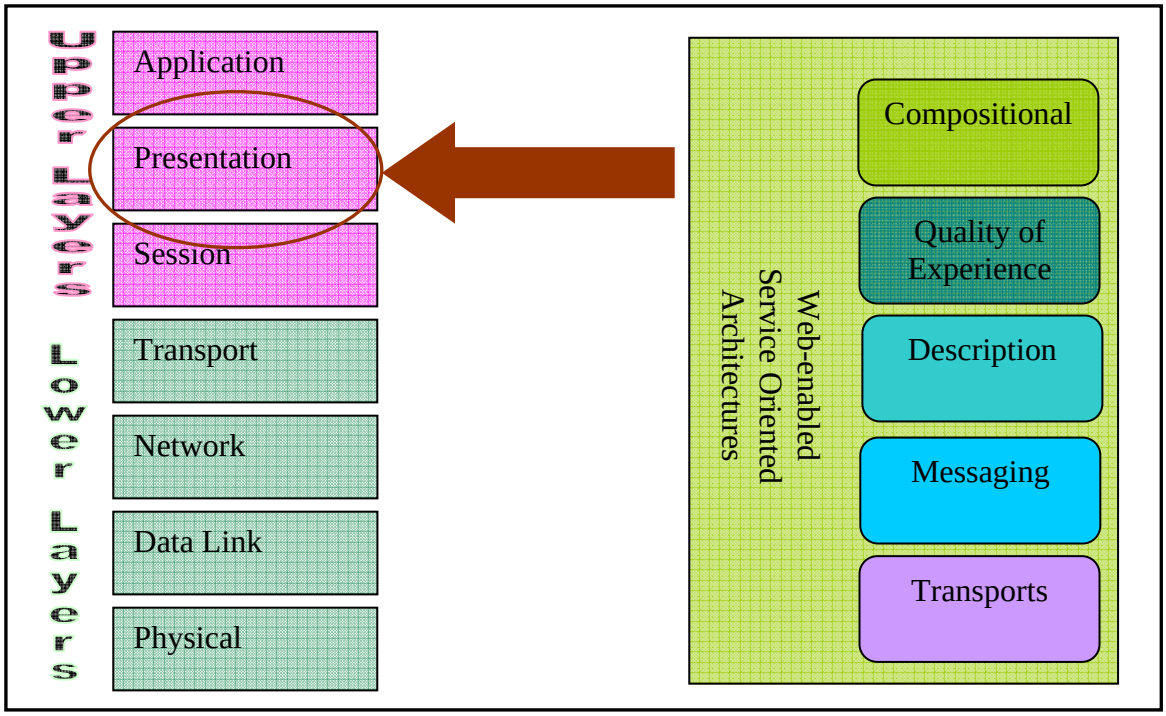

Figure 3. Graham's SOA"Stack"position in the OSI

Figure 4, below adopted from Erl shows the basic exchanges between services and a service registry. The diagram below is technology independent, however webservice standards have demonstrated a potential to fulfill the architectural requirements of a SOA. For example the webservice description language (WSDL) standard has meta-data about a service in a standard format such as a description of the

content and location. A registry standard such as Universal Discovery Directory Index (UDDI) can store service information in an open and standard format. Finally a standard method of data exchange of exists in standards such as Simple Order Access Protocol (SOAP). In summery a primitive webservice SOA is supported by three current standards; WSDL for service description, SOAP for messaging, and UDDI for service registry (Erl, 75).

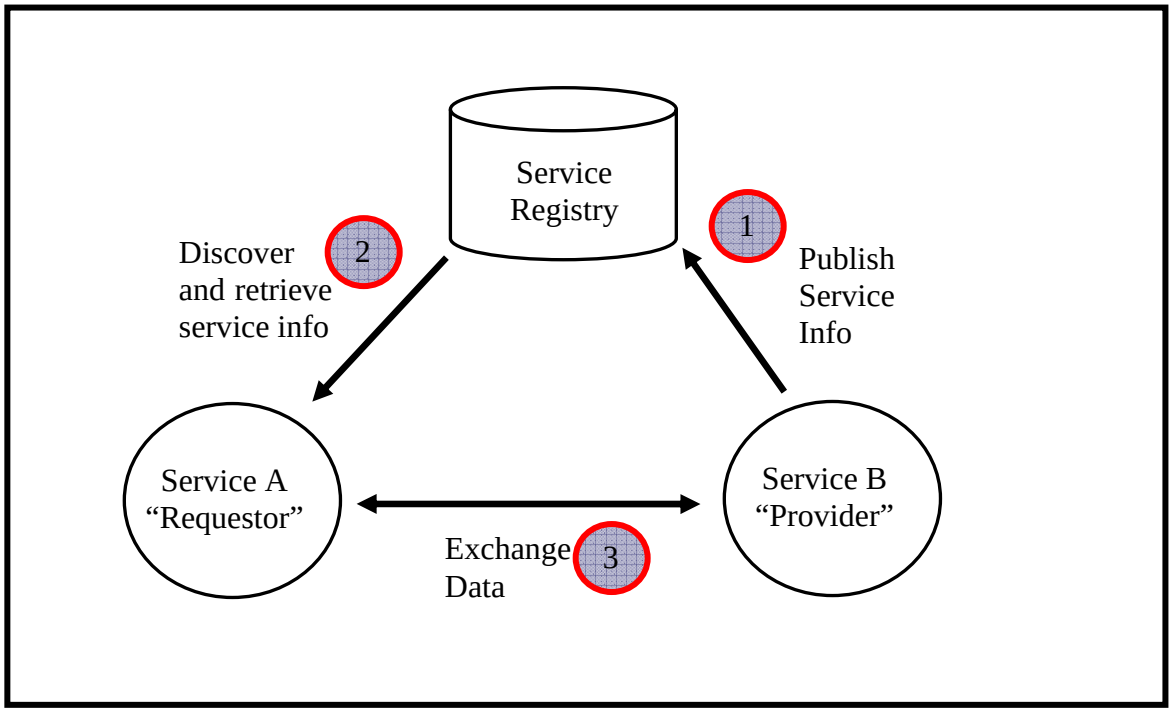

Figure 4. Basic SOA Relationships

Primitive SOA's provide interoperability improvements, but additional standards exist to address greater functionality. These standards include methods to orchestrate and protect services. Orchestrating services provides a method to chain services into new and more complex services (Graham, 549). In the prototype software, orchestration is employed to join the two raw data service's (or report-level) sources of data to a fusion engine which then delivers it to a new follow on service (or track-level). How it is employed is discussed in greater detail in chapter three. Although promising, not all of the technologies are widely adopted and mature. The more mature technologies such as

those in the transport, messaging, and to a limited extent the description layers of the interoperability stack shown below in Figure 4 as adapted from Graham's *Building Web Services with Java* (Graham, 27).

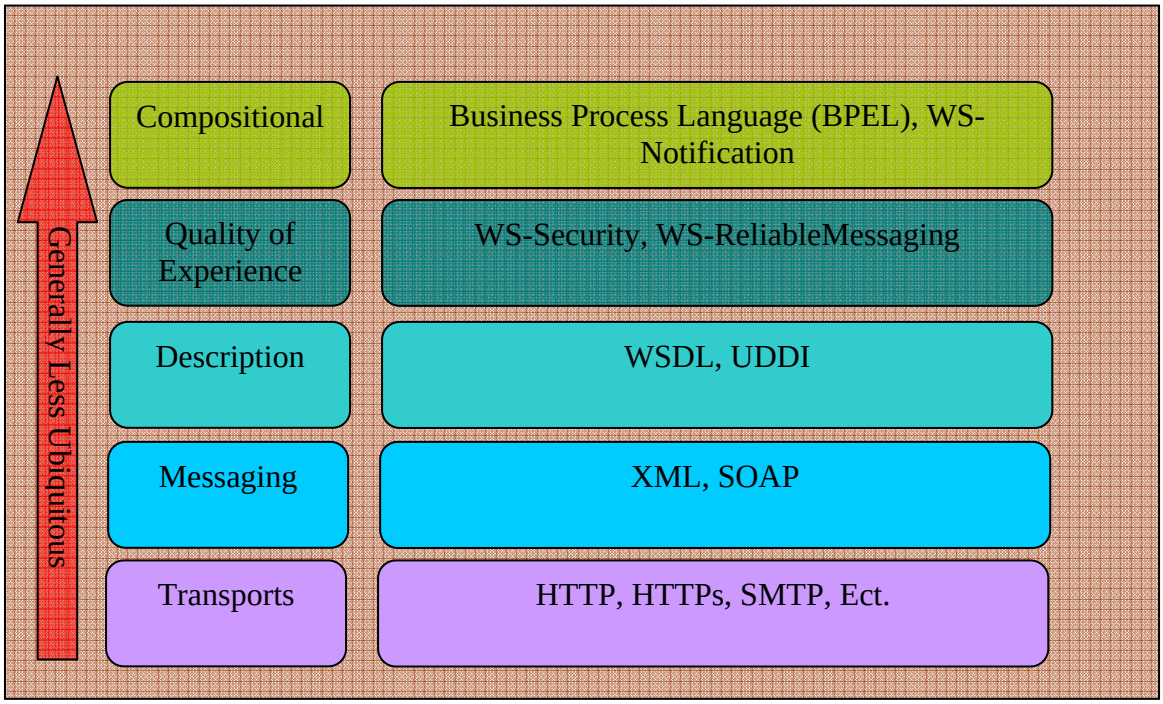

Figure 5. Graham's Web services interoperability stack

Examining the above stack, the lower levels are widely adopted by industry, while some of the advanced functionality layers such as security and Business Process Language are less mature and industry adoption is limited. HTTP, SMTP are ubiquitous standards and are the foundations of a number of technology solutions. XML, SOAP and WSDL are not yet ubiquitous but are gaining acceptance and the compositional and Quality of Experience types are still limited to early adopters.

The prototype software will reside on the standard SOA web-services platform similar to the one adopted by the MDA pilot, which currently is the sun application server or BEA Weblogic. The research began by migrating a "legacy" or non-SOA application developed by the author and discussed in greater detail in chapter three to a webservice SOA. This prototype employed the core services of discovery, orchestration and messaging discussed above to include Universal Discovery Directory Index (UDDI) and

Business Process Execution Language (BPEL) to accomplish the data fusion tasks. Additionally web-service standards such as Simple Object Application Protocol (SOAP) and Web Service Definition Language (WSDL) will also be employed in the development.

II. BACKGROUND

A. MULTI-SOURCE DATA FUSION BACKGROUND

In 1991 the Office of Naval Research (ONR) chartered a data fusion development strategy to guide investment in fusion technology. The research described the process, benefits and needs for data fusion. The body of work is now one of the authoritative guides on fusion and provides a good starting point to explain the concepts and challenges associated with multiple-source data fusion (Hall, 1-8). The ONR working group defined military-related data fusion as "the function of continuously transforming data and information from multiple sources into richer information concerning: individual objects and events; current and potential future situations; and vulnerabilities and opportunities to friendly, enemy and neutral forces"(ONR, 6). According to the processing definition, the purpose of data fusion is to take content from two or more data sources to promote better situational awareness for warfighters and hence make better decisions, within the Observe, Orient, Decide and Act (ODDA) loop.

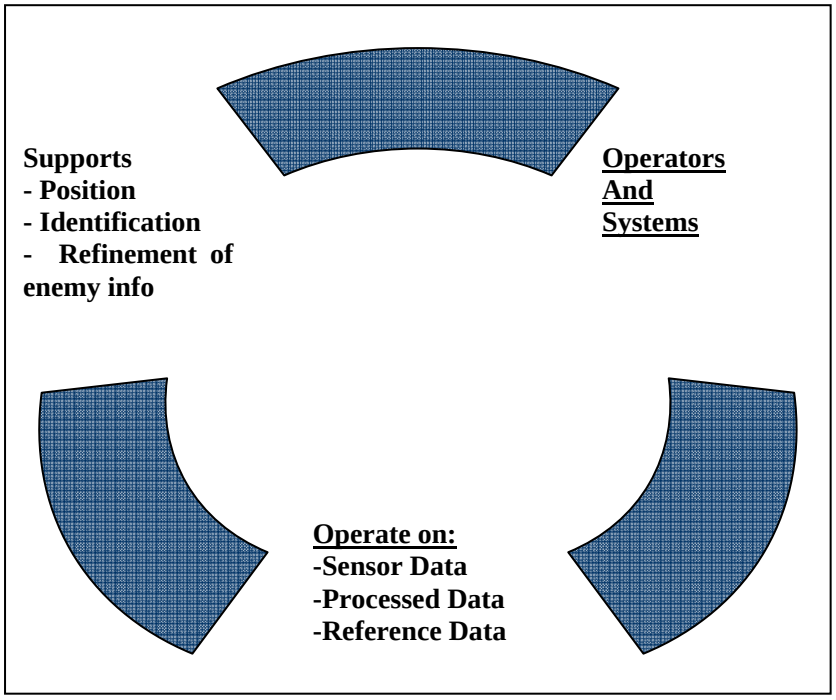

Figure 6. ONR Data Fusion Cycle

ONR defines a fusion cycle in which operator's direct sensors which provides data that is then fused to produce information about an entity of interest. Figure 6 shows that cycle. The relevance of data fusion in weapon and Command, Control, Communications, Computer, and Intelligence (C4I) system engineering has increased as the number, range and variety of sensors in weapons and C4I systems has increased. A number of factors have contributed to growing demand for fusion technologies to include:

- Improvements in technology supporting sensor ranges and sensitivity
- Growing number of sensors as more data types are desired by operators
- Increasing reporting rates due to proliferation of Internet Protocol based sensors
- Growing operator demand due to wartime requirements and Homeland Defense missions.

The ultimate military objective for data fusion is improved mission effectiveness. Data fusion contributes to mission effectiveness by reducing the amount of irrelevant data, improving information for decision-making and improved use of available weapons and sensors (ONR, 12). Data fusion priorities change depending on the position of the decision maker. A unit or individual ship operates in a low-volume, high-paced, limited-uncertainty environment, driving users to need increased accuracy to deal with the more limited choices that operators encounter. A strategic-level decision maker has a significantly greater volume of data, of greater latency and uncertainty than unit-level decision makers and hence requires greater inferences (ONR, 16). Hence we can make the argument that there is greater potential for benefit from fusion for strategic-level vice unit-level decision making. The requirement for strategic-level decision should be taken into consideration when formulating the data strategy and architecture for building a framework to support automated data fusion.

In 1986 the Joint Directors of Laboratories (JDL), which today is the Office of Naval Research (ONR), established a fusion working group to standardize terminology

and build a common model for data fusion (Hall, I7). The working group created a functional model with the four levels shown in figure eight below. The JDL panel defined four levels of fusion in the Data Fusion Domain: Level One Processing or object refinement, Level Two Processing or situation refinement, Level Three Processing or threat refinement, and Level Four for process refinement.

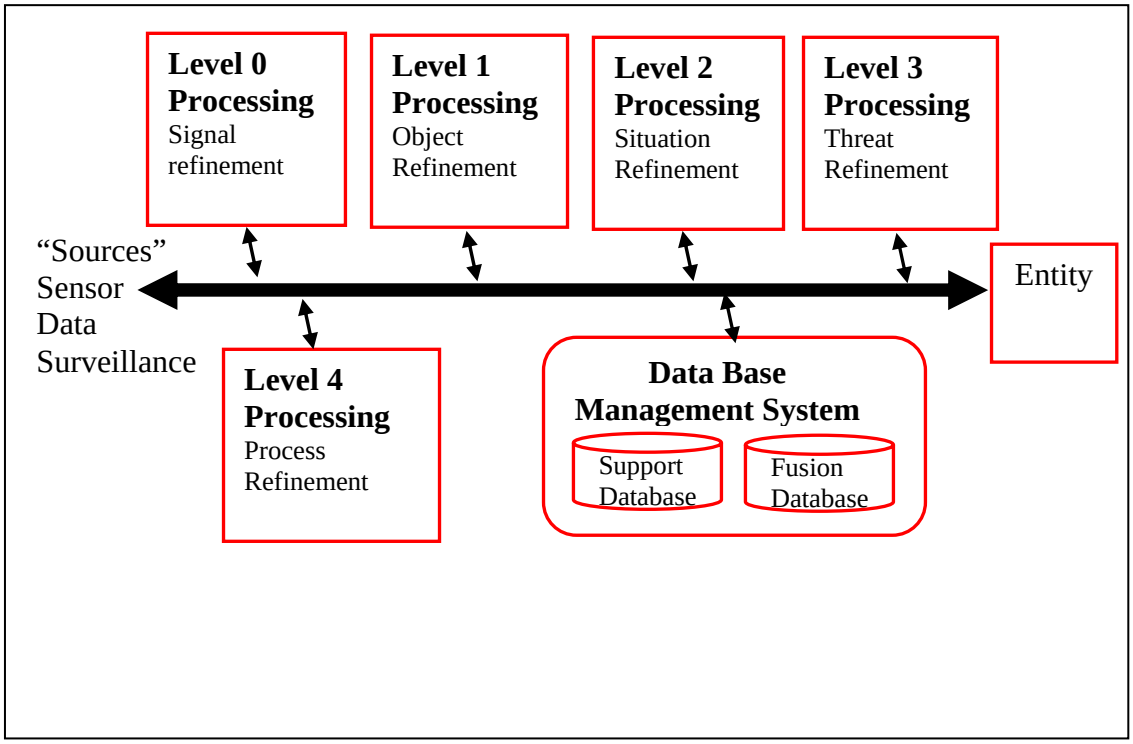

Figure 7. JDL Fusion Process

B. PREVIOUS SOA WORK IN THE ISR DOMAIN

The defense sector of the U.S. economy has already invested heavily in examining SOA as a method to increase efficiencies and interoperate with an array of legacy systems. Some view SOA as the next wave of "code reuse," where services are abstractions of classes in object-oriented programming. One central difference is the potential for services to be dynamically bound at runtime by an orchestrating entity (Erl, 448). Many DoD strategists believe a crisis is looming in the future as the complexity of systems and missions increases, while the number of individuals to complete those tasks

decrease (Bennette, 54). As the complexity and speed of the problems humans deal with increases, there is a tendency within the DoD to rely on software to solve those problems. Looking at the JDL fusion process above (Fig 7), we are looking more towards systems to automate Level Four processing.

For example, in the present conflict in Iraq, many disparate agencies are charged with the mission of reconstructing critical infrastructure and providing security. A complete view of the battle space is critical to providing visibility to these resources as well as traditional items of military significance. Resources in the case of reconstructing Iraq potentially include water treatment facilities, roads, electrical substations or other parts of a nation's infrastructure. However, present C2/common operational picture (COP) tools are normally track- or target-centric, limiting visibility of these resources because these items were viewed as tertiary parts of the battle-space. These non-military resources become increasingly relevant as a conflict progresses beyond combat operations to nation building or operations other than war (OOTW).

Furthermore, the conflict in Iraq demonstrates the requirement for military C2 systems to interact with a far wider source of content than ever before. U.S. military assets not only operate with the defense forces from other nations, but work with law enforcement agencies, intelligence communities, non-governmental organizations (NGOs) and private security firms in supporting the reconstruction and nation-building agenda in Iraq. In many cases the U.S. military provides quick-reaction forces and medical evacuation across the entire battle-space to numerous private security firms and coalition partners. Due to the classified nature of military position and planning data, sharing data between unclassified private C2 systems and military systems remains difficult. Beyond the security issues remain technical hurtles associated with the various proprietary military and commercial systems. The result of the limited data exchange architecture is cumbersome and inaccurate reporting, in addition to delayed reactions to events.

The preceding examples typify some of the wide spectrum of interoperability challenges posed by the diverse requirements for information on the modern battlefield. Figure 9 below represents the requirements overlap between domains. Future warfighter

support requires interoperability in a rapidly changing environment; an open and distributed architecture demonstrates potential to provide the architecture to address that requirement.

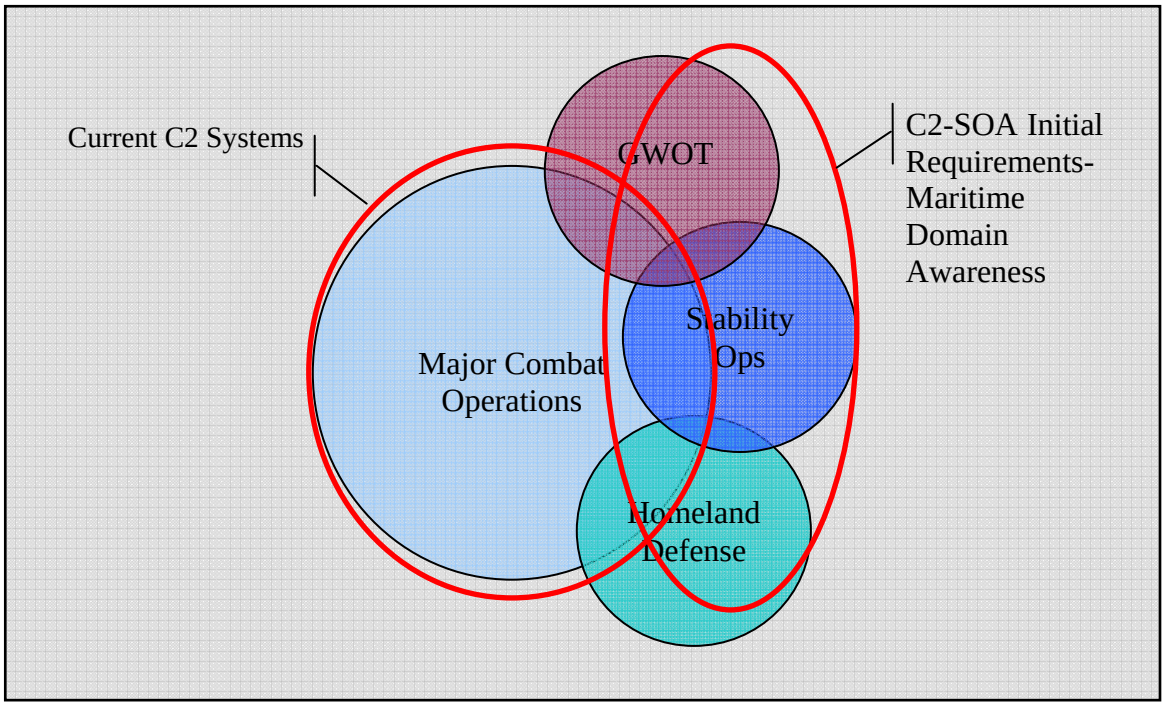

Figure 8. Chief of Naval Operations strategic focus areas with system overlay

C. COMMAND AND CONTROL/INTELLIGENCE SURVEILLANCE AND RECONNAISSANCE BACKGROUND

In the 1960's a transformational force enabler was born from the requirement to rapidly and accurately coordinate naval forces and their weapons systems over a wide area. That system is the Tactical Digital Information Link (TADIL), more commonly known as LINK-11/16 (Logicon, 3). Far advanced in complexity and potential of previous C2 systems, TADIL permitted the rapid allocation of resources to meet emerging threats or offensive requirements and dramatically increased the pace of operations. The TADIL Link today remains the premier air and surface warfare (SUW) fighting network, extensively used in recent combat operations in Iraq. Although

compelling, LINK-11/16 has some significant disadvantages over alternative systems today. These disadvantages listed below inhibit enhancements, increase cost, and discourage extensibility.

- Hardware specific (software is tightly coupled to hardware)
- Strongly typed software (no runtime enhancements)
- Tightly coupling: interoperability was difficult due to lack of common standards
- No convention for multilevel security (supports coalition warfare)
- Lack of common standards inhibited auto fusion and correlation of disparate data sources (TADIL format does not stem from open commercial standards)

Many of these were technological limitations associated with the hardware of the time, and the endurance of the system over the years is a tribute to the engineers who build it. The central tradeoff was performance versus extensibility. The code and hardware interface was clean and efficient in its use of memory and bandwidth, with extensibility being sacrificed for performance.

1. Emergence of PC-Based C2 Systems

In the early 1980s a growing personal computer market emerged from the existing mainframe industry. During that period the Navy began experimenting with PC-based tactical decision aids (TDAs). TDAs focused on automating computationally intense operations such as generating optimal search plans and had limited user acceptance due to the complexity of user interfaces and poor integration with existing C2 systems to provide the "true" environment. To address these shortfalls, the TDAs began interfacing with C2 systems via a passive tap to populate the TDAs with pseudo real-time pictures of blue and red force units.

As often happens with technology, customer use of TDAs rapidly departed from the system designer's original intent and began to be employed as a C2 system. The intuitive map, rudimentary routing (navigation voyage planning) capabilities and relatively easy to use interface stood in stark contrast to the existing link user interfaces. Along with the growing need for PC-based TDAs, a growing niche emerged for a PC and

Internet Protocol C2 tool that was portable to an ashore or afloat command center and in some cases operated at the theater level (vice link pictures that operate more locally at the tactical level) and could operate independently of a link architecture. At the core of this system is a database residing in volatile memory known as track management system (TMS). As user adoption increased, additional requirements emerged to support injection of national-level intelligence and surveillance data into a COP, further driving requirements away from a monolithic system towards a collection of systems operating under a common user interface.

2. The Common Operating Environment

In response to dramatic growth in adoption of PC-based C2 systems in the mid 1990s, the Navy led an effort to neck down the number of applications the warfighter had to touch and provide a common interface for a wide array of tool sets and data. The result of this effort is the Common Operating Environment (COE), based on the original Joint Operations Terminal (JOT) that migrated to the Global Command and Control System (GCCS) based family of TDAs and data sources consolidated into a single product line (Engel, 3). A number of dynamics were at play during this period that significantly impacted the present product line of PC-based C2 systems. These factors include:

- Reduction in hardware costs due to commercial market forces
- Reduction in DoD funding
- Reduction in number of defense contractors due to industry consolidation
- Dramatic increase in afloat bandwidth and PCs from the IT21 program
- Adoption of platform-independent programming languages
- Wide-scale commercial adoption of web-based hypertext protocol services

These factors converged on an acquisition community fielding calls for greater transformation and an increasingly technically savvy user group. The resultant product, GCCS-M COE, or specifically for the maritime user, GCCS-M (3.X), has a client-server architecture and an IP-based synchronization tool that maintains a multi-carrier strike group/theater-level COP. At its core remained a single database, Track Management System (TMS) and executed its requirement for theater-level COP by mirroring the TMS

database using a tool called Cop-Synch tools (Engel, 3). Figure 2 demonstrates the complexity of the COE and its wide user base.

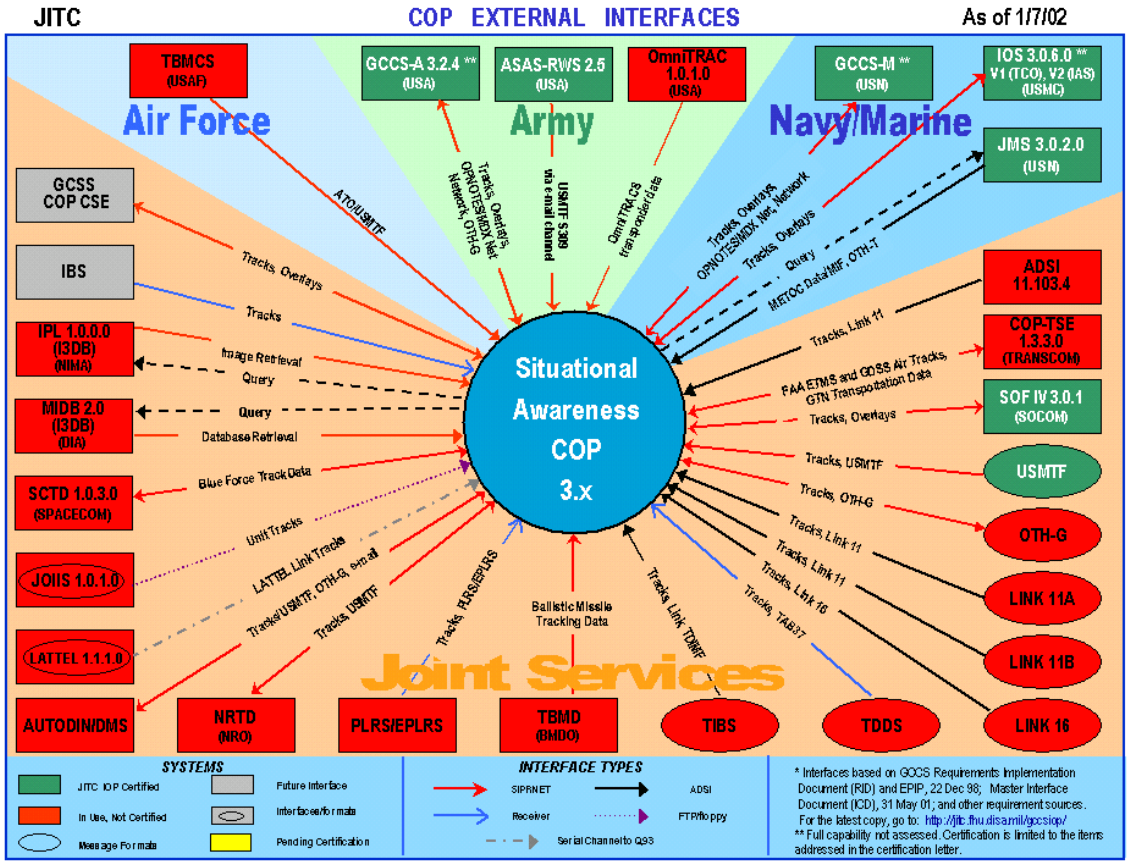

Figure 9. The GCCS Common Operating Environment Architecture

Presently an upgrade to the GCCS-M system is in the process of fielding. The GCCS-M 4.X upgrade improves user interfaces, makes the Clint code PC vice UNIX compatible and provides some web-based clients and access to TMS. Although an improvement in usability and porting to a new client hardware system was completed, no dramatic changes to the architecture were made. The system maintains TMS at its core, making integration of new services and clients slow, costly, and track-centric. To improve warfighter situational awareness the next generation system must break the present functionality into discrete standalone services that are dynamically discoverable and can

connect/disconnect post run-time. An example of an architecture that supports those goals is a Service Oriented Architecture.

D. POTENTIAL OF SERVICE-ORIENTED ARCHITECTURES

The commercial Internet has dramatically impacted communication and business processes in both the military and commercial world. However, the commercial world has different data models and requirements than a military user, prohibiting a one-to-one mapping of requirements. The military, and in particular the afloat user, maintain different quality-of-service (QOS) levels, an exceptionally wide array of data types, and a complex and rigid acquisition process. Additionally, the political factors associated with the use of public funds further impedes the process of system development and adoption, creating the present day lag between commercial and military systems.

There are numerous motivations for adopting a SOA in the C2 environment, but none of them are more compelling than extensibility. Extensibility for C2 systems is the ability to incorporate new data sources, clients and tools that manage and manipulate the data without recompiling the code. For example, how complex is the integration environment, how well documented are the interfaces, and what are the security hurdles imposed by the framework. In short, extensibility is the process of expanding the architecture to other sources of data and clients, and is a metric of how high the barrier of entry truly is for services to join the SOA. Extensibility in the past has often been a tradeoff with performance. However, there is less need for a tradeoff, as network bandwidth increases and microprocessors become more powerful.

Beyond extensibility, benefits of a SOA permit a restructuring of how C2 systems are acquired, by mitigating some of the roles a core integrating prime contractor filled and enabling greater competition. Additionally, resources can be made available for more tailored applications that work among the systems subscribing to the SOA. Some additional high-level requirements for the next generation COP operating on an SOA includes:

- Non-proprietary code for central communications methods and data models
- Industry-based standards

- Security
- Platform independence
- Interoperability with real-time systems (e.g., two-way interface with Combat Systems and Tactical data links “TADIL”)
- Support for IP-based collaboration

Along with the benefits comes greater responsibility on the government acquisition community and operators to manage the SOA. However a tangible improvement in capability will result as the barrier of entry is lowered and more attention is focused on the issues.

Presently numerous DoD agencies are adopting SOA as a vehicle for enabling their respective Net-Centric vision. Along with the other services, the Navy’s FORCEnet vision has also embraced the SOA concept to increase speed of command and reduced costs due the extensible nature of SOA (SECNAVINST, 5). In the late 1990s, Navy Research and Acquisition commands recognized the potential flexibility Publish and Subscribe (PUBSUB) SOA offered C2 systems and began research efforts to translate those visions into reality. As network technology matured, Message-Oriented Middleware (MoM’s) emerged as a key enabler for SOA along with Simple Access Object Protocol (SOAP), over Hyper Text Transfer Protocol (HTTP) networks, as well as emerging Enterprise Service Bus (ESB) technologies.

E. SERVICE ORIENTED ARCHITECTURES BACKGROUND

Numerous definitions of Service Oriented Architectures exist. One definition (Barry, 2004) is simply “a collection of services” that can pass data and communicate with each other. Paul A. Moore defines SOA’s more rigorously, with ten design-time and eleven runtime characteristics. Developers often examine the quality and maturity of the components and design and look directly down to the XSD (XML Schema Document) of the messages being passed to see if they are structured with extensibility in mind. A factor of extensibility can be affected by the granularity of the XSD. For example XML elements like <Latitude> / <Longitude>, or <Location>, or perhaps the

data is binary. In the former the data element can almost directly be used by a program without much effort. In the latter, it may imply that further parsing of the data may be required. Which path is utilized depends on the system requirements, where the burden of further development is shifted from one developer to another (e.g., database developer to visualization developer). Hence we see two central metrics for quality of SOA's emerging: **Extensibility of messaging** (both in messaging systems and content of message) and **Independence of systems service components**.

Barry (2004) states that communication may be in the form of either simple data passing or could involve two or more services coordinating some activity. SOA is not a new concept (Barry, 2004, p.1). The first widely adopted SOA is DCOM (Distributed Component Object Model [Microsoft]) or Object Request Brokers (ORBs) based on the CORBA specification (Barry, 2004, p.1.). A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services (Barry, 2004). For services to be accessible a method is needed to make a connection to access the service. One technology currently employed to connect services is Web Services. Web Services refers to the technologies that allow for making a web-based connection such as Universal Description, Discovery and Integration (UDDI) for discovery, Simple Object Access Protocol (SOAP) for passing an XML document, Business Process Execution Language (BPEL) for orchestration and XML for content. A service is the endpoint of a connection, with a type of underlying computer system that supports the connection offered.

As discussed above, independence of system service components is another level of analysis that compares an SOA is in terms of how "componentized" their services are. In essence the metric measures how well-defined, discrete, and self-contained the services are which translates to greater ease of integration with other systems. For example, are the services modular enough in the sense of what would it take to replace the service with a better service? Ideally, SOA services are very well-defined and self-contained, can be easily replaced and follow a type of plug-and-play, drop-and-replace structure. In summary the following top-level attributes of an SOA are (Barry, 2):

- Well-defined components
- Self-contained components
- Connectivity between internal and external processes

In the next section we will discuss some of the foundation technologies of SOA in greater detail.

1. Extensible Mark-up Language (XML)

XML is emerging as the "content" of choice for DoD applications. Poorly suited for use over previous networks, due bandwidth limitations, XML is gaining traction now that bandwidth has increased and system extensibility and connectivity are becoming a higher priority. Until recently, XML was not a viable medium for the DoD until processor speeds and bandwidth reached a level where performance met the levels of preexisting legacy systems. Presently, processor speeds and network capacity are reaching the point where costs in performance are worth the gains in extensibility. XML is an extensible version of Hypertext Markup Language (HTML). Designed to permit developers to define the model or schema desired dynamically and share the model without reconfiguration. In many ways, HTML can be thought of as a small version of XML focused on display, while XML is concerned with more varied uses of data. XML is however very structured and unambiguous permitting its use in DoD systems.

The compelling reasons for DoD adoption of XML stem from its platform independence and its non-proprietary nature. In Command and Control systems, common XML schemas have tremendous potential to create a common language between sensors, weapons and visualization clients. Of greater importance is the capability for expanded fusion, correlation and automatic data management. XML allows the proliferation of fusion services throughout the net to automatically compare disparate data from far flung sources to build a better Common Operating Picture. For example data from a National ELINT source with thousands of reports can be published to a topic that is both discoverable and "subscribeable" by a fusion service that is also subscribing to a TADIL track feed for auto correlation. The product of the fusion service is then published to a

new topic each of which can be subscribed to by a visualization client, event manager or tactical decision aid. Although fusion algorithms can be built that can read the *tab37 (standard)* format for the ELINT and the *MTC (LINK)* format for the TADIL data, the service becomes extremely unique and tightly coupled. By utilizing XML as the common language the task for fusion service developer becomes simpler and can therefore focus on better algorithms vice reading the data and exception handling.

Beyond fusion, XML permits a standard language for visualization clients to integrate simplifying design for the presentation layer as well. The Navy should however be wary of anyone selling the "definitive" model for everything. XML schemas of that size become poor performers with much of the content of limited utility to the subscriber. A one size fits all schema runs contrary to the philosophical underpinnings of XML. Better to use XML provided translators (XSLT) than attempt to create an "Uber Schema." From an anecdotal perspective the Office of Secretary of Defense, National Information Infrastructure office sponsored a Horizontal Fusion track working group to develop a "track schema" to satisfy all the participants. In reality it satisfied no one and was mocked from inception by developers working on the project. Another schema project is C2IEDM, even more ambitions than Horizontal Fusion, this organization born from the concept to define DoD "entities" in a relational database schema, the organization has gained traction as a XML schema provider. Supported by the North American Treaty Alliance (NATO), C2IEDM does not map seamlessly to XML, leaving a lot to the imagination of developers and may not deliver the promise of a DoD wide XML definition for Command and Control. Developers need to continue to build tools that are as schema independent as possible and keep an ear to the ground for emerging XML standards to use them when they make performance and extensibility sense.

2. Message Oriented Middleware (MoM)

In more ways than other services, the Navy is a "messaging" culture. Stemming from the asynchronous nature of the afloat environment, where ships and submarines were only available to communicate during limited periods of time (e.g. in-port, within visual singling range, at periscope depth). This culture, and in reality a limitation that exists to this day to a lesser degree, drove many of our communication systems and our

very fighting doctrine. Many of our systems are serial feeds, and broadcasts that do not assume quality of service on the receiving end. As a culture, the Navy values concise, directive communications in the form of "message traffic" whose authority is validated by the complexity of getting the object "on the wire" and the penalties for misuse of the broadcast. Potentially, this makes Navy Command and Control systems good candidates for Message oriented Middleware.

In contrast to developers making their own stove-piped system from physical transport medium to application, JMS continues the trend of greater development abstraction. Here the abstraction is instead of programmers writing their own network socket code for both producers and consumers of data, MoM's permit developers to place the data on "Topics" and "Ques" allowing any subscriber to access the data required. Hence DoD specific developers time is spent on developing the defense specific system versus the software to communicate between application. No capability is free of tradeoffs in resources or performance and MoM is no different. MoM requires vendor specific servers, server software and costs in network performance. The major two vendor specific MoM's (.net and JMS) are discussed below.

Java Message Service (JMS) is a standard created by Sun Microsystems for enterprise wide messaging. Incorporated into Sun's Java 2.0 Enterprise Edition, (J2EE) the technology entails establishing an enterprise wide messaging bus via a *JMS Server* for applications, databases and event managers to share data while still running independently. For example, in contrast to building a business wide "single application" much like *SAP*, Sun envisions accounting programs, marketing/sales programs, inventory management, etc, all developed and operating independent, but sharing data in a standard asynchronous manner. Referring to the figure below the JMS service is intended for application info sharing and not just exchanges between clients and databases.

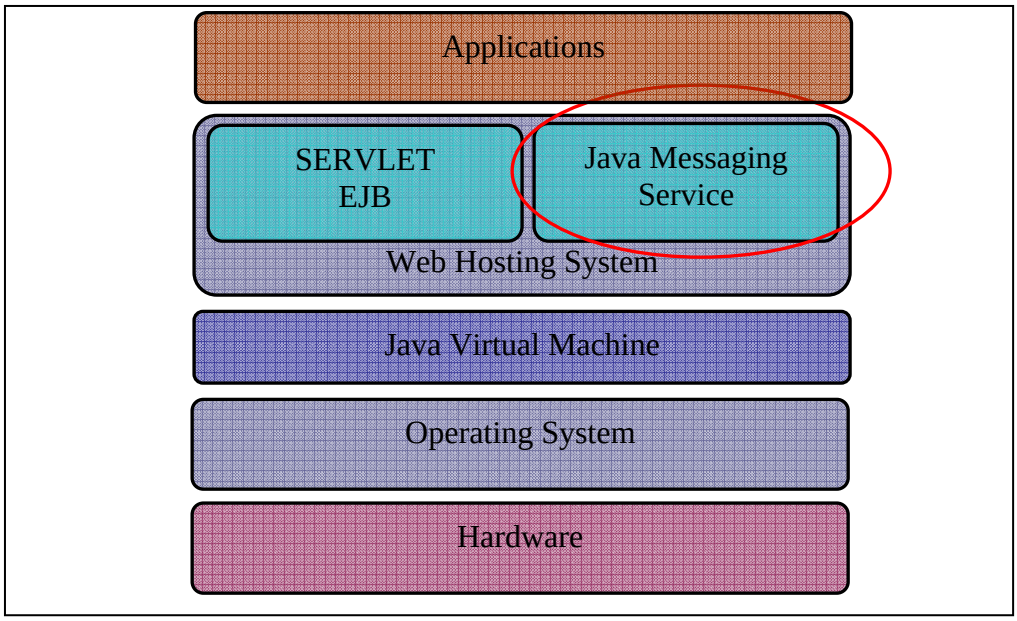

Figure 10. J2EE Technology Stack

The architecture of the JMS Enterprise bus is not functionally a client server model (all messages must process through a JMS message server, but do not have to go through the data source central services) but conduct communications in a pier to pier fashion. The JMS Server has *agents* that run in conjunction with system end points that manage the communication in a decentralized manner.

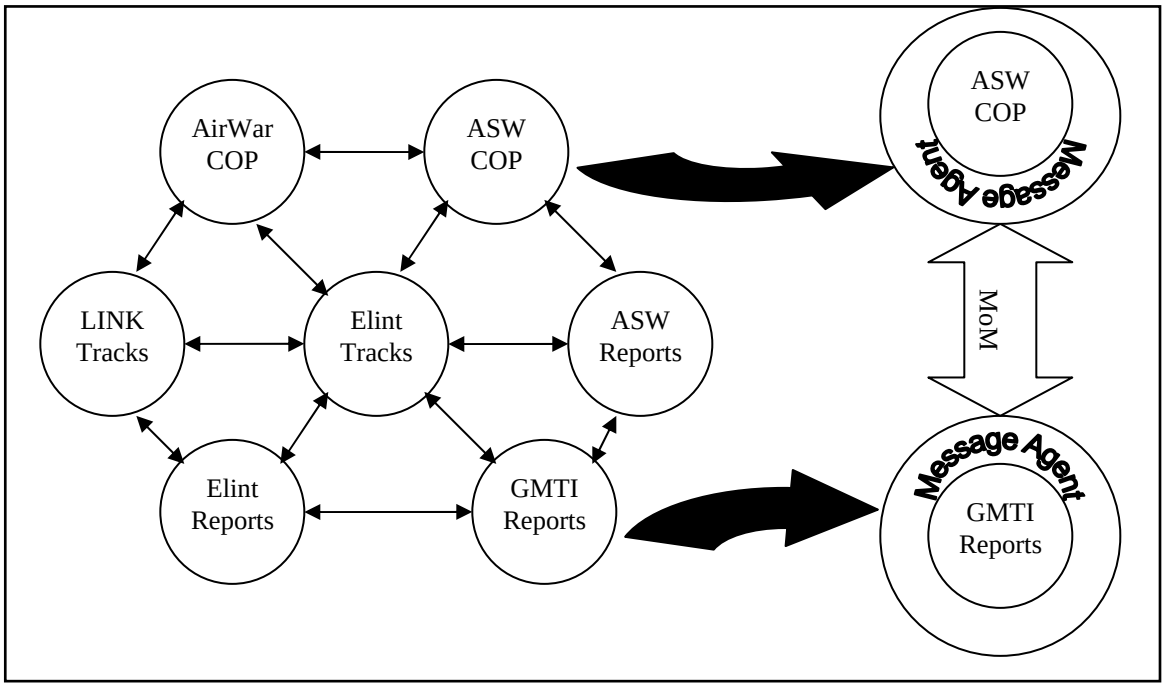

Figure 11. Reference JMS Afloat Architecture

In the diagram above we have an example of a reference "Navy" JMS instance. In the model each service or endpoint utilizes a MoM, message *agent* that handles defining the destination, priority and quality of service required. Some of the key attributes that make JMS a compelling solution for DoD and in particular Navy afloat Command and Control system network connectivity include the following attributes:

- Supports both JAVA and non-JAVA message payloads
- Supports both intra-service and service to service communications
- Supports asynchronous data transfer between nodes

Although Microsoft leads the Personal Computer Operating System, healthy competition still exists on enterprise level software solutions such as Message oriented Middleware. Comparing the two systems technologies stack, .NET has similar mechanisms as JMS for message exchange (Shiel, 2). For example, .NET supports a wide range of the quality of service attributes to match JMS and in particular has superior

development tools and strong Multilanguage support. The downside to .NET is its single vendor approach, questionable security and limited multi-database support. The diagram below demonstrates the relative position of messaging on the .NET stack, similar to the JMS model. How .Net handles messages and provides a distributed environment is similar and most of the discussion above applies.

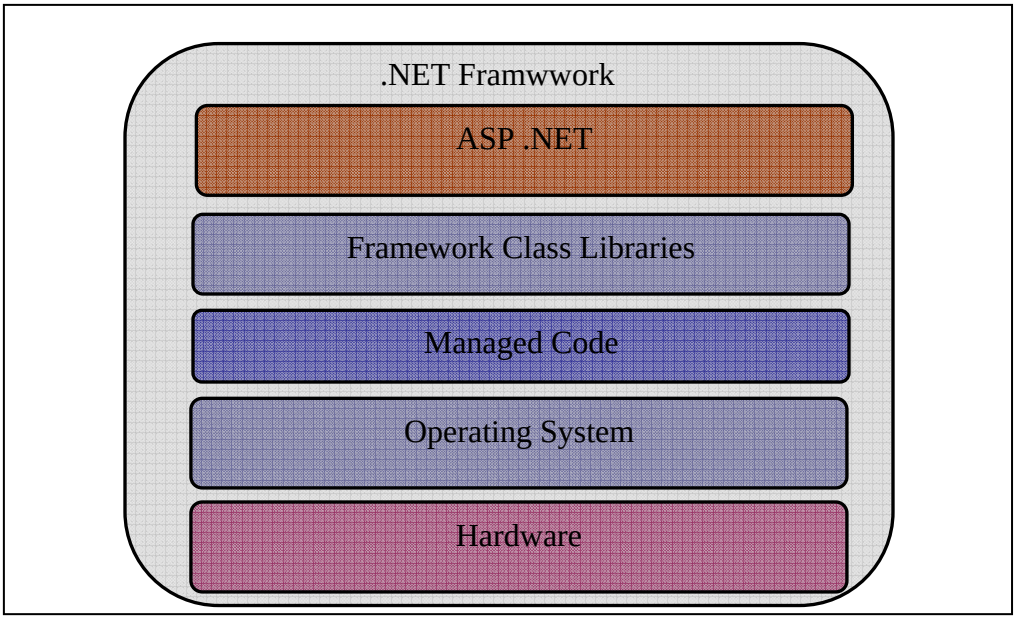

Figure 12. Net Framework Technology Stack

3. Orchestration and Discovery

Although the messaging and XML data standards lower the barrier of system-to-system integration, the real benefits to be accrued of having independent services are realized when services can be reorganized to respond to changing requirements after runtime, which is one of the promises of discovery, and strung in a string of services as provided by orchestration. At present there are two standards for orchestration that are competing with each other: Business Process Execution Language (BPEL) and *WS Orchestration. Discovery or XML based registries are provided by both ebXML standard from OASIS and Universal Discovery Directory Index (UDDI). The dissertation prototype employs a BPEL engine provided by Sun Microsystem.

THIS PAGE INTENTIONALLY LEFT BLANK

III. TRICKLE-UP AND ZONE SOFTWARE DESIGN PATTERNS

Christopher Alexander defines design patterns as a method to "describe a problem which occurs over and over" (Gamma, 2). Relating the constructs Alexander was referring to in the construction industry, Gamma et al; apply the framework to object-oriented programming, by articulating a series of repeatable software patterns. As discussed earlier, SOA is in many ways an abstraction of object oriented analysis and designs (OOAD) and hence can benefit from object-oriented pattern constructs as well as new ones. Additionally, since SOA is different from object-oriented programming, one would expect patterns to emerge to capitalize on SOA (Buschmann, 30).

Gamma describes four essential elements of a pattern: pattern name, description of problem, description solution, and consequences from applying the pattern (Gamma, 3). Following the methods proposed by Gamma, we will describe the two patterns developed in this dissertation, in the following section using the pattern essential elements. Each pattern discussion includes; intent, motivation, applicability, structure, participants, collaborations, consequences, implementation, sample code, known uses and related patterns. Additionally, Gamma provides an organizational structure to patterns into three general purposes (i.e, creational, structural and behavioral) and scope (class or object). In this section we will discuss two new software design patterns proposed in this research, and employ Gamma's techniques, with some modifications to align with SOA constructs, to describe them.

A. TRICKLE-UP SOFTWARE DESIGN PATTERN

At first glance, the trickle-up design pattern appears as a simple n-tier construct, with three levels (i.e., atomic, refined and entity) that generally map to the knowledge management constructs of data, information and knowledge. The difference however, stems from the aggregate nature between the levels and the relationships between the objects or source services and the processing algorithms. The trickle-up pattern has its roots in a data model developed by Dr. David Engel, known as the *ISR Three-Tier Model,* while working on the eXtensible Tactical C4I Framework (XTCF) at the Space and

Naval Warfare System Center in San Diego (Engel, 3). The data model was an artifact in the XTCF system design document to introduce developers to a construct where data sources, fusion logic and presentation systems where no longer tightly coupled in a single system, but resided in a loosely coupled SOA. In this dissertation the three-tier model is explored beyond the original construct, migrated to a pattern and beta-tested in a prototype. The following sections describe the *trickle-up* pattern using Gamma's structure, with modifications provided by Erl to relate the pattern as a SOA design pattern vice an object-oriented design pattern.

1. Pattern Name

The name of a pattern should convey as much as possible the intent of the patter, allowing communication of the topic to speed understanding (Gamma, 3). In the case of this pattern, the phrase *trickle-up* refers to a concept where water would slowly flow up against the pull of gravity to nourish a plant. The phrase reflects how actions at the bottom of a layered structure can aggregate over time to impact higher layers of the structure. Trickle-up is normally associated with two strategies unrelated to software. The first is a strategy the New York Police Department (NYPD) adopted to address crime by aggressively prosecuting petty crimes with the hope that a general "lawfulness" will reduce all types of crime. The second strategy refers to a method adopted by relief-aid agencies to encourage small businesses with micro-loans to tackle chronic poverty. The trickle-up pattern follows this analogy with the concept that data from the bottom is fused with other data to migrate up to create information of greater value.

2. Pattern Intent

Intent provides a concise description of the pattern. The *trickle-up* pattern provides a multi-tier abstraction in which both source data and fusion logic are independent services and defined in a manner to enable runtime binding to facilitate an aggregate approach to control and monitoring services software data management.

3. Pattern Motivation

Control and monitoring software systems can use remote sensors to collect data about the environment, analyze it and make a report to a consumer. Traditional systems employ tightly coupled methods to pass the data, limiting the reuse of the content and creating a classic "stovepiped" system. Utilizing web-based SOA technologies and standards, one can construct a framework in which individual data managers can autonomously provide persistence and be employed by any consumer to include a service that performs analysis on the content. This supports the concept of low or "loose" coupling which reduces the dependencies between modules to improve flexibility in a software system (Erl, 37). The central motivation for using the pattern is flexibility. Flexibility in this case is the ability to add, remove or modify both sources of data and methods of analysis as the system matures.

Consider a system for a medical enterprise with a number of hospitals. To optimize operations the medical enterprise has various types of information systems, such as medical diagnostic monitoring systems. A diagnostic monitoring system has numerous sensors that report on the health of a patient, such as blood pressure, hearth rate and temperature. As new sensor types become available, the content can be made available to any number of other systems, such as diagnostic rules-based engines or algorithms for taking readings from any number of sensors to recommend treatment. These systems can link with the pharmacology department to ensure the wrong drugs are not administered, or a personnel manning system to ensure the professionals with the right set of credentials are on station when required. This complex system of systems benefits from the agility of the *trickle-up* pattern, allowing the adding of atomic, complex or entity sensor information service in a correct category and allowing the pairing of data services with a diagnostic service after runtime. The services can now be changed together with an orchestration engine dynamically in a process. Next we will talk about how this pattern is applied.

4. Pattern Applicability

Use the *trickle-up* pattern when

- You want loose coupling between sensor and application services, with both types of services being configured during runtime in response to changes
- You want multiple “fusion engine” or other processing-logic services to connect at runtime, allowing optimization of the fusion process by employing the best service components (e.g. algorithms) at that point in time
- Your target software is a distributed system and has a range of data objects that vary in complexity and higher order information is derived from combining data from various sources.

In the next section we will discuss the patterns structure.

5. Pattern Structure

The *trickle-up* pattern articulates a structure where individual services data are combined to improve awareness. The basic building block of the pattern is an object manager service (OMS). The OMS takes the input from a sensor, then both normalizes and saves the data. The OMS provides two general methods to obtain values: one static and another event-based. The former may report results based on values provided, while the latter provides a continuous updating of subject of the OMS. An eventing based update is a continuous subscription like an audio stream or broadcast, where the consumer remains connected to the OMS and awaits updates. The OMS is autonomous and maintains its own history. Additionally, the OMS is registered in a discovery service making it available for use by various customers. Figure 13 below details the OMS. The figure uses SOA service notation style adopted by Erl (Erl, 497).

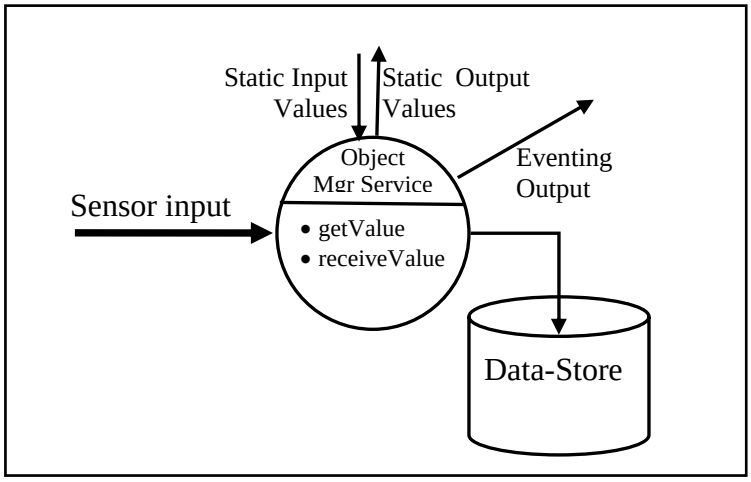

Figure 13. Basic Object Management Service

The second type of building block in the *trickle-up* pattern is the data fusion service (DFS). DFS utilizes the data content from multiple Object Management Services (OMS) for analysis purposes. DFS has two inputs (one for each OMS) and an output that provides the new objects created for input into a new OMS. The DFS does not have its own persistence and does not maintain a history of its actions, but rather provides its output to another OMS. The method executed by the DFS is performFusion; this method takes the values provided by the OMS inputs and performs the fusion, creating a new output which in turn flows to the output OMS. Figure 14 below diagrams the DFS.

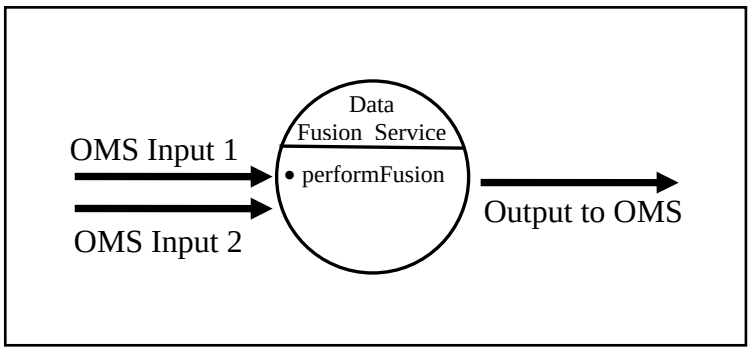

Figure 14. Data Fusion Service

With the two main building blocks defined, we can now assemble these blocks to specify a complete *trickle-up* pattern. Applying a brick and mortal analogy, the OMS is a brick and the DFS is cement, in a building. There are three general types of OMS "bricks": atomic, complex and entity, which are joined using the DFS cement. Assembled in a reference framework, the *trickle-up* pattern looks like a pyramid with atomic data at the bottom, complex types in the center, and entities at the top. As shown Figure 15, the arrows point upwards because data from the lower level objects migrates upward as the complexity of the data increases. Some additional relevant concepts regarding the pattern are:

- The pattern does not require that all data "start" at the atomic level, but can in fact join the pattern at the appropriate level of complexity.
- The pattern does not require that all DFS's must subscribe to two OMS's. Many circumstances exist where a DFS may subscribe to a single OMS with different parameters.
- Some OMS's may never need fusion.

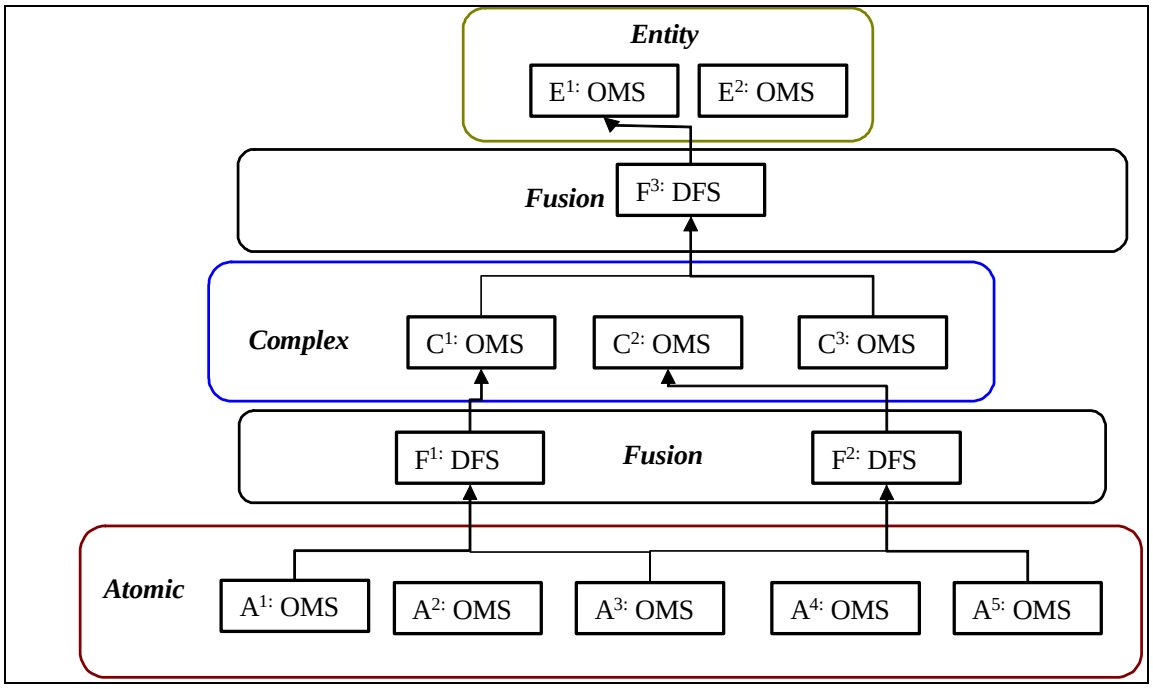

Figure 15. Trickle-up Software Design Pattern

In Figure 15 above, "A" annotates an atomic OMS, "C" annotates a complex OMS and "E" annotates an entity OMS. The term "F" on the diagram represents a DFS.

To further explain the trickle-up pattern, a UML structure diagram is provided in Figure 16. The pattern comprises two "object classes", the OMS and DFS, described earlier in this section. Two broad categories of OMS exists; composite which has both complex and entity type objects and atomic. The DFS has two broad categories, based on Bowman's defined types of fusion algorithms, relationship and attribute (Bowman, 2-4). The OMS has a data store, but the DFS does not. Note that an OMS can exist without a DFS, however a DFS requires at least one input OMS and one and only one output OMS. In the diagram, a DFS ingest of content from an OMS is called preprocess, and the output of the DFS to another OMS is called a postprocess. Additionally, the diagram details how a DFS does not postprocess down the pattern, in other words, a postprocess cannot write to an atomic level OMS described earlier in this section.

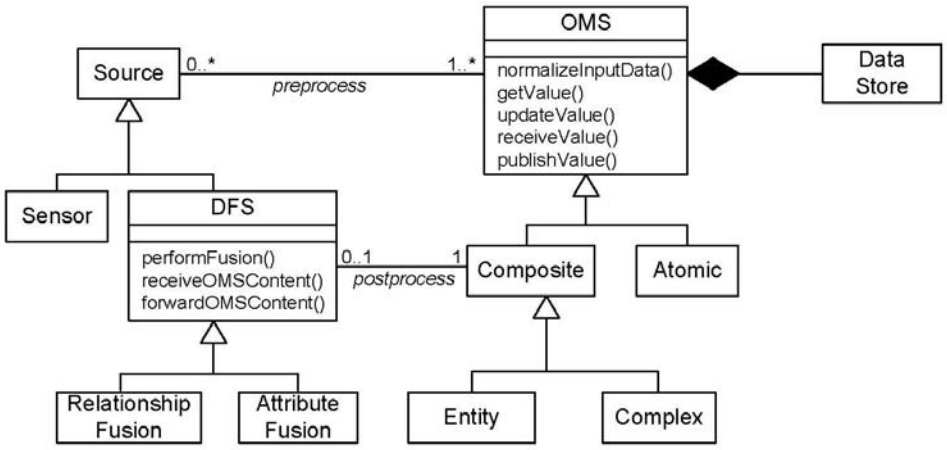

Figure 16. UML Diagram of OMS and DFS

6. Pattern Participants

Participants are the actors in the pattern. For the trickle-up pattern the participants include:

- Atomic Level Object Management Service
  - Defines interface from sensor/source
  - Implements the persistence of atomic objects
  - Defines the interface for static atomic object requests
  - Defines output of static atomic objects
  - Implements the reporting of event-based atomic objects
- Complex Level Object Management Service
  - Defines interface from sensor/source
  - Implements the persistence of complex objects
  - Defines the interface for static complex object requests
  - Defines output of static complex objects
  - Implements the reporting of event-based complex objects
- Entity Level Object Management Service

- Defines interface from sensor/source
- Implements the persistence of entity objects
- Defines the interface for static entity object requests
- Defines output of static entity objects
- Implements the reporting of event based entity objects

- Data Fusion Service
  - Defines the inputs from partner OMS
  - Implements the fusion algorithm
  - Defines the output to partner OMS

7. Pattern Collaborations

The trickle-up pattern has a number of collaborations between its consumers, sensors, data managers, and fusion services. The two main collaborators are the OMS and the DFS. Using a sequence diagram format, we can map the relationships of the pattern in a similar fashion to that described in Larman (Larman, 141). First we will show the OMS sequence followed by the DFS.

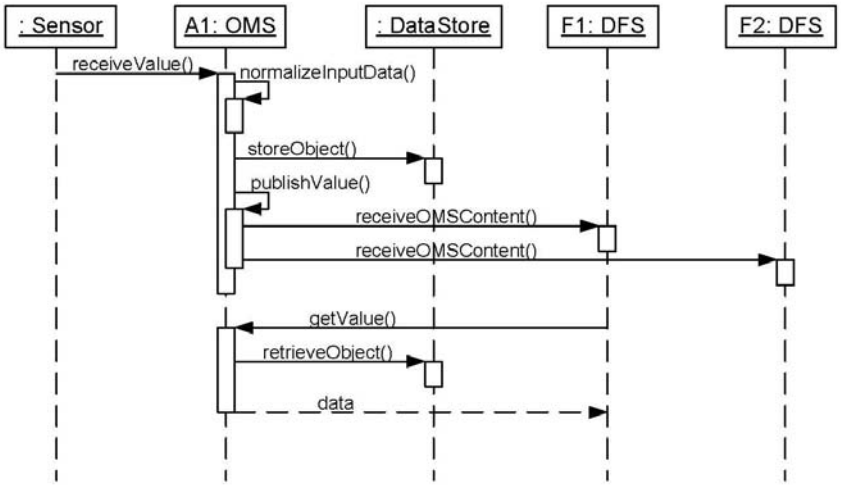

Figure 17. Collaborations for OMS

The OMS collaborations execute the following

- The source inputs to the OMS
- The OMS saves the object
- The OMS sends the object to a eventing consumer (optional)
- A consumer such as a DFS or other, requests objects and has them returned

For the Data Fusion Service (DFS) we have a similar set of collaborations..

- Sources are sent to the DFS
- DFS creates the object
- DFS fuses the sources and makes a new object
- DFS publishes the new object

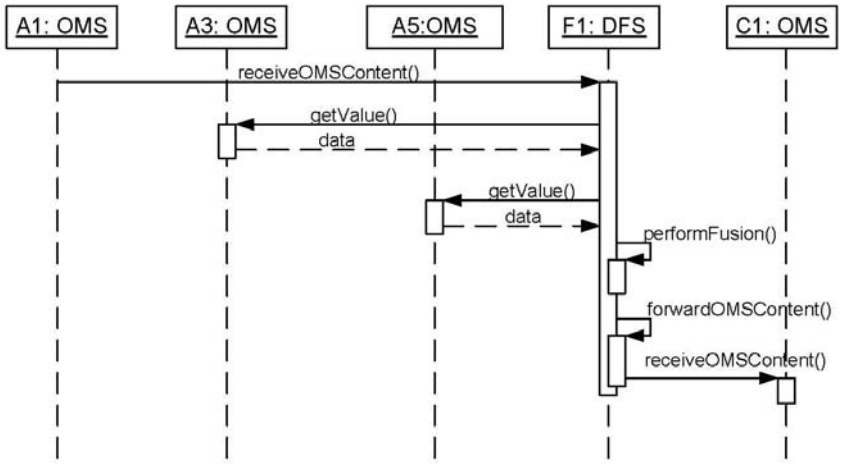

Figure 18. DFS Collaborations

Note the sources of the DFS are OMS's and the new objects are posted to a new instance of an OMS. The significance of this is to stress that a DFS cannot serve a customer directly and must provide its output to an OMS.

8. Pattern Consequences

The trickle-up pattern is a fundamental technique for reusing fusion and data source code in a SOA. The pattern presents a view of the control and monitoring software domain sources into two main categories: data managers and fusion processors. Within the data managers, three categories of complexity are defined by complexity of data: atomic, complex, entity. Software reuse is enhanced by separating the fusion logic or DFS, from the standard "building blocks" or object management service (OMS). The pattern specific consequences include:

- The pattern requires common data-schemas and definitions to support "late binding" and orchestration of the OMS and DFS modules in a SOA
- The sensor/source content has a potential value from aggregation (combining one object with another from a different period or type).
- The type of content lends itself to a atomic, complex, and entity construct
- The system has overhead associated with the flexibility it provides

9. Pattern Implementation

Although the trickle-up pattern has some applicability in a traditional object-oriented system with interoperability between systems facilitated with an API, the pattern is geared towards a Service Oriented Architecture framework. Since web-service technologies are the present method to enable a SOA, the *trickle-up* pattern discussed in this dissertation is focused on SOA methods. The trickle-up pattern can be implemented using Representational State Transfer (REST) or an Enterprise Java Bean (EJB) framework. The REST is usually a servlet-based system and the EJB employs a SOAP message with an XML based message body. The EJB construct is aligned with the web-service SOA-enabled business process execution language (BPEL) methods and is employed in this research prototype. The REST method in the research here is executed using Java Servlet technology and is used to provide the mapping service overlays via a

keyhole markup language (KML) interface. These efforts are described in more detail in the following prototype section. Referring back to figure 13, the EJB or Servlet provides methods to process static inputs and eventing or static outputs. The dynamic output can be accomplished with any number of eventing methods such as Java Messaging Service (JMS), DDS for more real-time systems or WS* eventing. I changed this to 3rd order paragraph

The following Java code shows an example of an OMS. The basic OMS below uses the EJB classes which are called @Stateless and @WebService. The class uses a Java Array list structure and returns values from a database.

```
@Stateless()
@WebService()
public class basicOMSeJB {
    Database database1 = new Database();
    QueryDataSet queryDataSet = new QueryDataSet();

@WebMethod
public List<object> processBasicOMSeJB(String parameterfirst, String
parameterSecond) {
     returnparamOne = queryDataSet("SELECT * FROM DB
WHERE  parameterfirst, parameterSecond");
   returnparamTwo = queryDataSet("SELECT * FROM DB WHERE  parameterfirst,
parameterSecond");

  return = returnparamOne,returnparamTwo
}
```

The second set of java code below show a potential Data Fusion Service (DFS) implementation. As in the OMS, the java Enterprise Java Bean framework is shown and @Stateless and @Webservice implements the EJB. Here no persistence is provided beyond the temporary array list. The DFS takes inputs from two OMS and conductFusion executes the DFS specific algorithm.

```
@Stateless()
@WebService()
public class DataFusionServiceFusion {

@WebMethod
public List<combined> isDataFusionService(List<object> objbatchOne,
List<objectTwo> objectbatchTwo){
```

```
    List<combined> fusebatch = new ArrayList();
    fusebatch = ConductFusion(objbatchOne, objectbatchTwo);
return fusebatch; }
```

The prototype software description in the chapter VIII goes into greater detail of the implementation for readers who would like to examine a reference implementation. Additionally, the reference implementation shows the data structures the OMS and DFS would employ and the orchestration software employed to bind them.

10. Pattern Known Uses

The *trickle-up* pattern is not employed in any production systems to date that the author can find; however some aspects of the concept was used by the Space and Naval Warfare System Center San Diego's (SPAWAR-SD) efforts in eXtensible Tactical C4I Framework (Engel, 4). XTCF used the pattern constructs of individual object services and data fusion services and demonstrated the porting of fusion logic to a basic SOA construct; however orchestration was not a focus of XTCF and BPEL was not employed.

11. Related Patterns

Trickle-up utilizes a number of object-oriented patterns. The *strategy pattern*, where algorithms are encapsulated supports the DFS construct (Gamma, 315) and the *composite pattern* supports the broad structure of the OMS as a hierarchy (Gamma, 163). Trickle-up is often conjunctive with the Zone pattern described in the next section. Trickle-up provides the foundation data sources for consumption and further processing by the zone pattern. The next section will describe the relationship between zone and trickle-up patterns.

12. Pattern Categories

The trickle-up pattern is a structural and object-based pattern. Structural refers to a type of pattern that focuses on the composition of objects (Gamma, 10). An object pattern focuses on relationships between objects (Gamma, 10).

B. C2-ZONE SOFTWARE DESIGN PATTERN

In contrast to the trickle-up pattern, the zone pattern is a behavior-oriented pattern. Behavior patterns are defined by Gamma as methods to characterize object iteration and distribute responsibility (Gamma, 10). Object iteration and responsibility distribution in a SOA framework describe how services cooperate to perform a task. The zone pattern provides developers of control and monitoring systems a view of the relationship and roles between a consumer of "authoritative" content, a "participant" in a control and monitoring system, and the sources of content. The pattern makes a distinction between a participant and consumer in a zone. A consumer is provided the content the zone manager approves, while a participant is considered a trusted part of the zone and has access to the entire view. The difference between a participant and consumer parallels a magazine printer relationship with its fellow writers and readers, in that the content available to those who work at a magazine (participant) is different than the finished product provided to magazine subscribers (consumers).

1. Pattern Name

Merriam-Webster defines *zone* as "a region or area set off as distinct." The zone patter provides for the separation of responsibility for communities of interest. Communities of interest are a group of people or entities with common interests and a need to collaborate (DoDCIO, 4). The term zone also conveys the authority and responsibility to accomplish a task, as in a "zone defense" in sports.

2. Pattern Intent

The zone pattern articulates a scalable structure to dynamically manage the three broad control and monitoring software relationships of source, participants and consumer.

3. Pattern Motivation

A fundamental challenge in control and monitoring system development is to decide where to place the analyst and processing node in distributed system architecture, how to manage the updates to content, and how to manage relationships with other participants in the domain. In a distributed control and monitoring system, the number of

remote sensors, interactions and business logic can change considerably over the lifetime of the system. The *zone* pattern supports flexibility in the positioning of the processing node in the system topology and the resources the node needs to accomplish the task. Additionally, the pattern supports the concept of separating the automated sources of information and the role a human analyst may make in validating or updating the content, along with the notion of ownership of the content.

Consider a national air-traffic management system. The topology changes over time as new airports are added and air traffic increases. Not only are sensors such as radars and ground stations for aircraft transponders added to the system, but processing stations are added shifting responsibility for managing a section of airspace. For example, an area in a country has had minimal traffic and one facility managed the airspace. But a new airport was built and air traffic increased beyond the capacity of the single station and the region was divided into two sections. The responsibility has shifted along with the resources to manage the airspace. The *zone* pattern supports the shifting responsibilities and resources requirement into a distributed control and monitoring system. In a military construct you can see how the agility is even more critical to support dynamic wartime requirements.

4. Pattern Applicability

Use the zone pattern when

- You want a loose coupled binding between content sources, participant and consumer systems.
- Your system is a distributed control and monitoring system for which expected changes in the system topology are significant enough to counter expected performance challenges from loosely coupled software technologies.
- Your system nodes have accountability constructs associated with them.

Next we will discuss the pattern structure.

5. Pattern Structure

The zone pattern articulates a cellular-like distributed structure, with nodes managing content and relationships. The basic functions are: persistence, sensor feed data normalization, data exchange/synchronization management, and consumer management. The *zone* pattern consumes the content from the trickle-up pattern and can have its output posted back to the appropriate tier of the trickle-up pattern (entity-level in most cases). Figure 19 shows a single zone element.

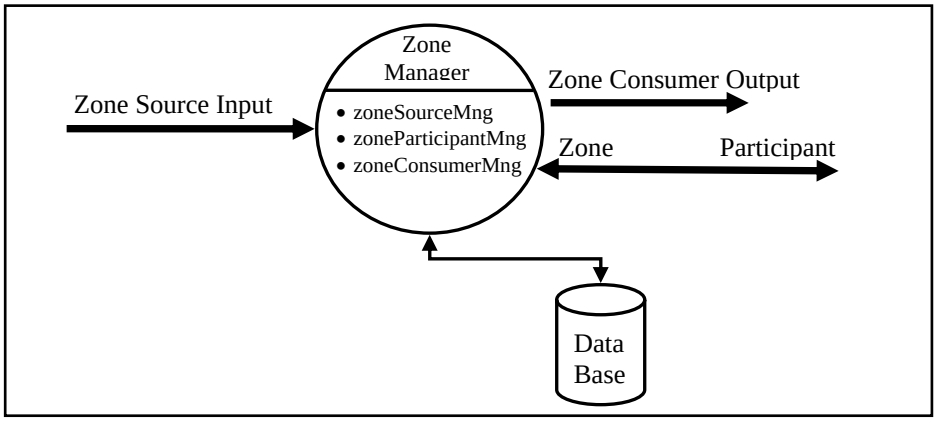

Figure 19. Zone Pattern Structure

The zone pattern is a SOA construct and the interfaces described above are enabled by loosely coupled interfaces and XML-based formats. The zone manager mimics many of the functions of the OMS discussed in the trickle-up pattern, in addition to both consuming and providing content for the trickle-up pattern. The Zone Manager is comprised of three general subsystems relating to the main interface categories shown above: Zone Source Manager (ZSM), Zone Consumer Manager (ZCM), and Zone Participant Manager (ZPM).

The Zone Source Manager (ZSM) manages the interface between the zone and its sources, in most cases a trickle-up pattern Object Management Services (OMS). The ZSI keeps track of any additions or deletions of sources and normalize the data to input into the persistence. Figure 20 shows the three actions the ZSM manages for adding and deleting sources, as well as normalizing and mediating the data from source to the persistent store data model.

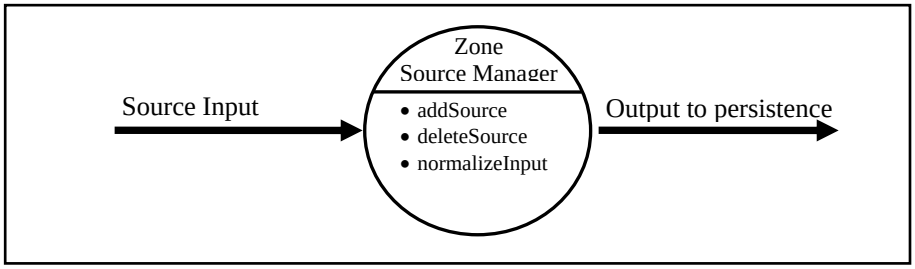

Figure 20. Zone Source Manager

The Zone Consumer Manager (ZCM) component of the zone manager manages the consumers of the data in its authorized form. The ZCM output can be consumed by other zones or other systems in a similar fashion as that used in the trickle-up OMS. Figure 21 shows the ZCM.

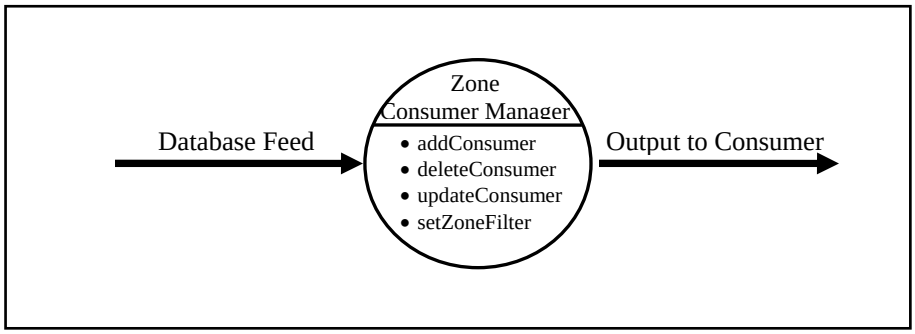

Figure 21. Zone Consumer Manager

The three main functions of the ZCM are adding, deleting and updating a consumer. Note that the ZCM does not synchronize or take inputs from the consumer, but provides the content from the zone. This is a critical distinction between the ZCM and the Zone Participant Manager (ZPM) module. The consumer is just that, a consumer. The ZCM should however provide a robust capability to keep the consumer updated despite changes in the network connection and support the loose coupling required by the pattern.

The ZPM module provides the methods to share the authoritative data with partners who assist in the management of the content. This is an “un-edited” or “raw” view of the content vice the “finished” product provided by the ZCM. The ZPM has similar functions to the ZCM but with the added responsibility of allowing a partner to

contribute or change the content. The ZPM has three general functions: adding, deleting and updating participants in the zones, as shown in Figure 22.

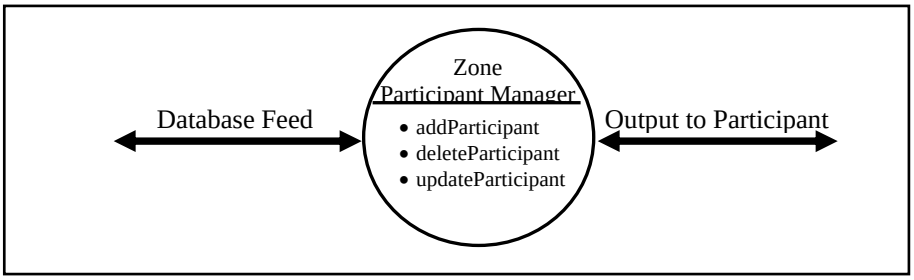

Figure 22. Zone Participant Manager

Another way of looking at a participant is as a member of the zone and inside a single zone. We will close the discussion of the zone pattern structure with an examination of how multiple zones are structured.

As discussed previously the zone pattern creates a cellular structure where individual zones comprised of sources, consumers and participants divide responsibility and accountability by zone. Figure 23 shows an instance of multiple zone interactions. The "O" stands for an OMS, the "P" stands for participants, "C" stands for consumers, and "Z" stands for zone manager. Also note that two zones can subscribe to the same OMS and provide to multiple consumers. Participants are unique to a specific zone.

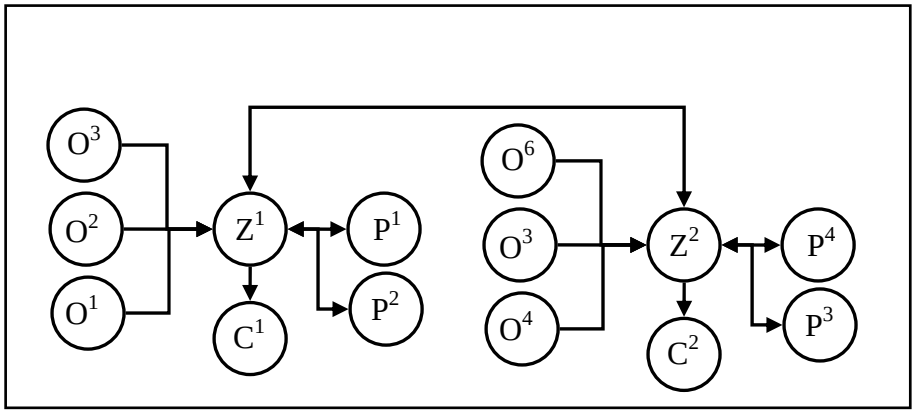

Figure 23. Multiple Zone Structure

Figure 24 shows a UML diagram of a zone detailing the relationships between the zone manager, OMS, data store, consumer, and participant. A zone can have any number of OMS, consumers and participants. A zone has a data store and can communicate with another zone or exist in isolation. A zone both consumes the content from none or many OMS, and can populate a trickle-up pattern as a single instance of an OMS. A zone can have none or many participant or consumers; however a participant or consumer cannot exist without a zone.

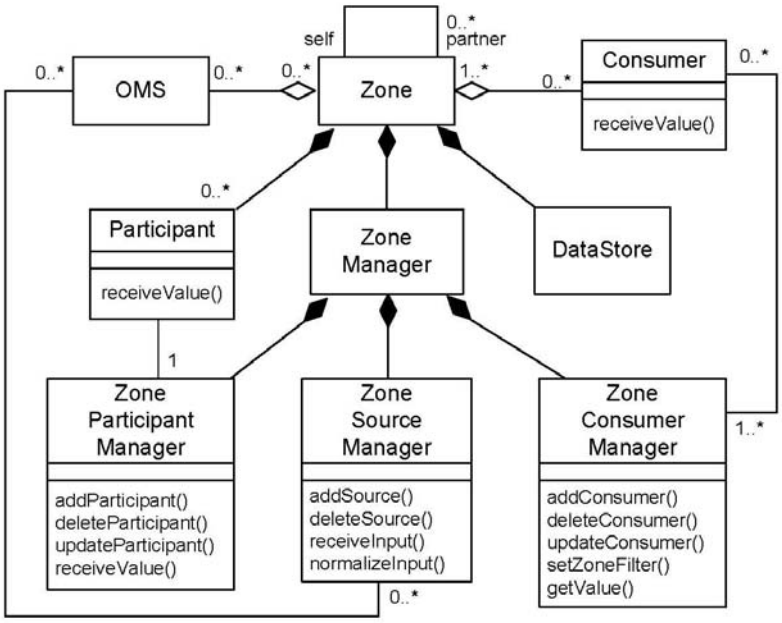

Figure 24. Zone UML structure diagram

6. Pattern Participants

Participants are the actors in the pattern. *Zone* pattern has the following participants. The OMS performs as described in the trickle-up pattern discussed in the previous section. Additionally, a zone would appear to an outside consumer with similar attributes as an OMS.

- OMS
  - Defines interface from sensor/source
  - Implements the persistence of objects

- Defines the interface for static object requests
- Defines output of static objects
- Implements the reporting of event based objects

- Consumer
  - Receives event based object updates as directed by the zone
  - Receives static based object updates as directed by the zone
- Participant
  - Synchronize complete content from Zone to Participant (participant is a mirror view of zone content)

7. Pattern Collaborations

The zone pattern has a number of collaborations between the data sources it consumes, the zone participants, and zone consumers. Similar to the trickle-up explanation in the previous section, we will use a sequence diagram to document the collaborations. The first collaboration shown in Figure 25 is the OMS.

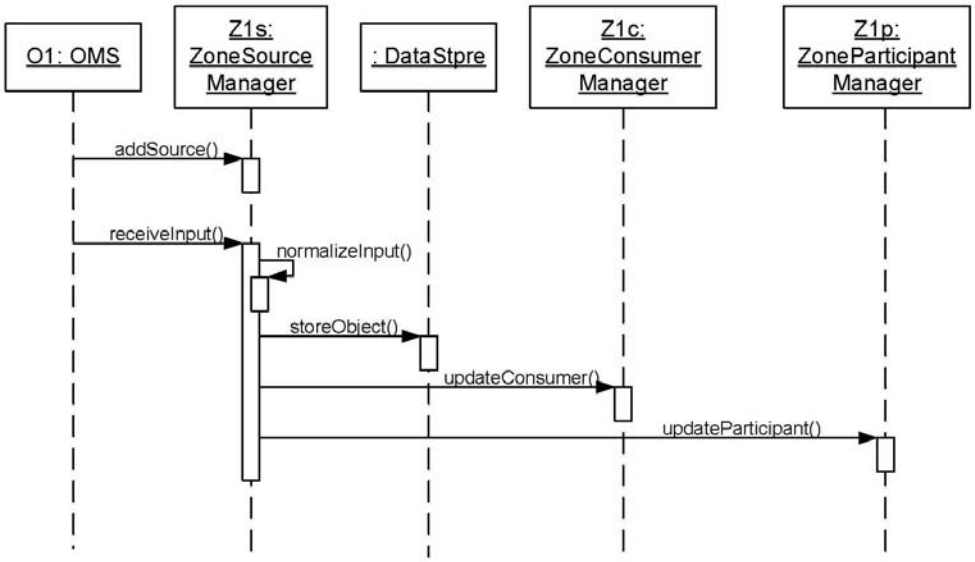

Figure 25. OMS to Zone Collaboration

Note the Zone can take input from any number of OMS. The Zone source manager then normalizes the data to match the format of the zones persistence or schema.

A Consumer is provided content from a zone. The zone manager coordinates access to only the desired and approved content. The interface between the zone and the consumer of its content, is shown in Figure 26. Note a zone can have any number of consumers and a consumer can either be another zone or some other type of consumer.

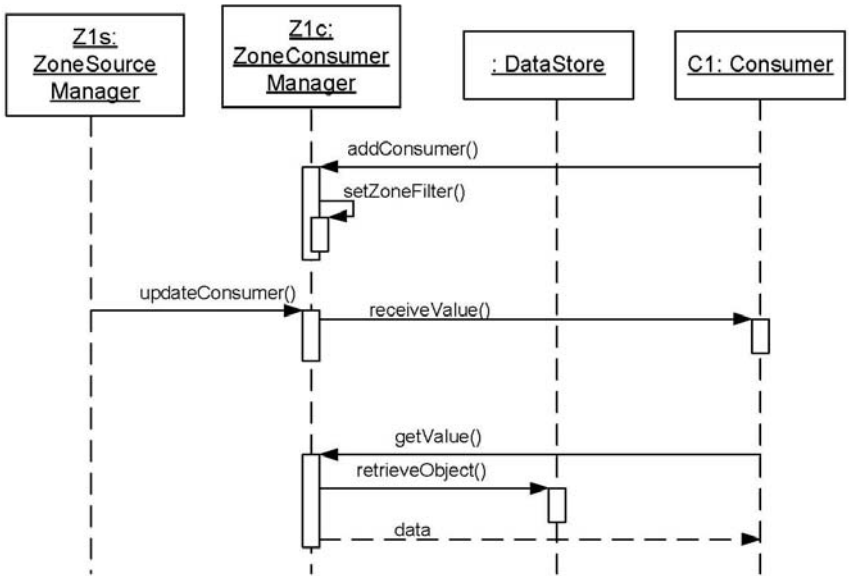

Figure 26. Zone Consumer Collaborations

For the Participant the collaboration involves two-way synchronization. A participant can add objects to the zone and needs to remain synchronized with the zone, as shown in Figure 27. Note the removal of normalization because the content is already normalized from the data store and the two-way synchronization. A Zone can have any number of participants, but a participant is mapped to only one Zone.

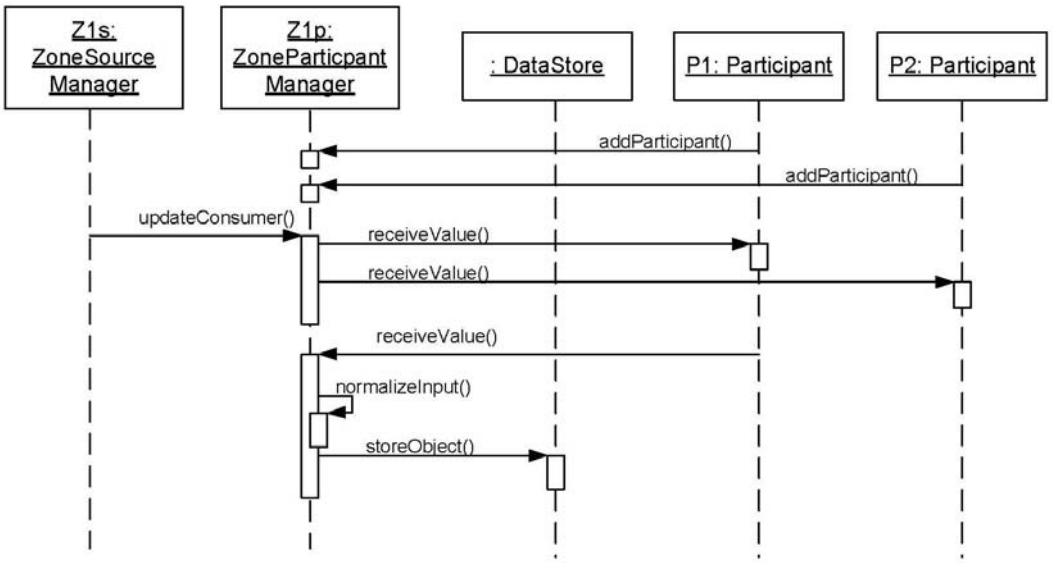

Figure 27. Zone Participant Collaborations

8. Pattern Consequences

The zone pattern is fundamentally a method for creating a scalable and extensible control and monitoring software management nodes in a SOA. The zone pattern supports extensibility by supporting the construct of multiple sources or OMS inputs to the zone. This zone pattern supports scalability by supporting a cellular like structure that allows for specific consumer definition of desired data and rapid establishment of partner zones. Two main differences exist from a system built in the zone pattern and traditional systems. Firstly, the zone pattern articulates a difference between consumers and participants in the zone. Secondly, the zone pattern articulates flexibility in the ultimate topology of the zone structures. This manifests in systems that can respond to changes in the environment by swapping out sources, partners and participants. This agility is facilitated by SOA loose coupling, but comes at a cost of performance and the overhead of using SOA based technologies and bindings. Additional consequences of the pattern include:

- The pattern requires a common data-schema and definitions to support the "late binding" of adding sources and consumers

- The desired topology of the target system lends itself to dynamic-authority constructs

9. Pattern Implementation

Similar to the trickle-up pattern, the zone pattern has some applicability in a traditional object-oriented software framework; however the pattern is really targeted towards a web-enabled SOA. Advances in database technologies as well as XML based transports and messaging facilitates interoperability. Higher order languages support the application aspects of the pattern such as administration functions, but the interoperability technology should remain as loosely coupled as possible. Similar to the OMS, a messaging service can provide the eventing-based data while a static feed via REST technology can provide the static content.

10. Pattern Sample Code

The following source code snippet contains a XML schema definition (XSD) file for the data exchange between nodes. Similar to the trickle-up pattern, Java can parse and read the data. However the critical element needed to employ SOA technologies is a standard definition of the object being passed. Note the types and names of the data types being passed.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0"
        targetNamespace="http://ZoneEJB.sun.com/
        <xs:element name="arg0" type="xs:string" minOccurs="0"/>
         <xs:element name="arg1" type="xs:string" minOccurs="0"/>
        </xs:sequence>
        </xs:complexType>
        <xs:complexType name="processZoneResponse">
        <xs:sequence>
         <xs:sequence>
            <xs:element name="ZoneObject" type="xs:string" minOccurs="0"/>
            <xs:element name="sigLat" type="xs:double"/>
            <xs:element name="sigLon" type="xs:double"/>
           </xs:sequence>
          </xs:complexType>
        </xs:schema>
```

The snippet above is a trivial example with elements containing only name, latitude, and longitude. A true XSD would have a significantly larger data set. The transport used to exchange the data would most likely be a Java Message Service with a SOAP message and the above type format in the SOAP body.

11. Pattern Known Uses

Although components of the pattern exist in any command and monitoring system, the author could find only limited use of the pattern. As in the trickle-up pattern, some of the zone pattern constructs are employed by the software engineers at the Space and Naval Warfare System Center San Diego (SSC-SD).

12. Related Patterns

The zone pattern augments a number of object-oriented design patterns to support its SOA descriptions. It primarily relates to the trickle-up pattern since trickle-up is its data source.

13. Pattern Categories

The zone pattern is primarily a behavior object type pattern as defined by Gamma, et al. (Gamma, 10). Behavior refers to the way zone patterns interact and distribute responsibility, while object refers the relationship between the objects or services in a SOA framework (Gamma, 10).

IV. APPLYING TRICKLE-UP SOFTWARE PATTERNS TO MDA

An unlimited number of ways exist to define and describe the metadata associated with military C2. In this research we apply the trickle-up pattern and apply them to MDA missions. Applying the patterns to MDA, the universe of C2 data is divided into three tiers as shown in Figure 28: reports (atomic), tracks (complex), and entities.

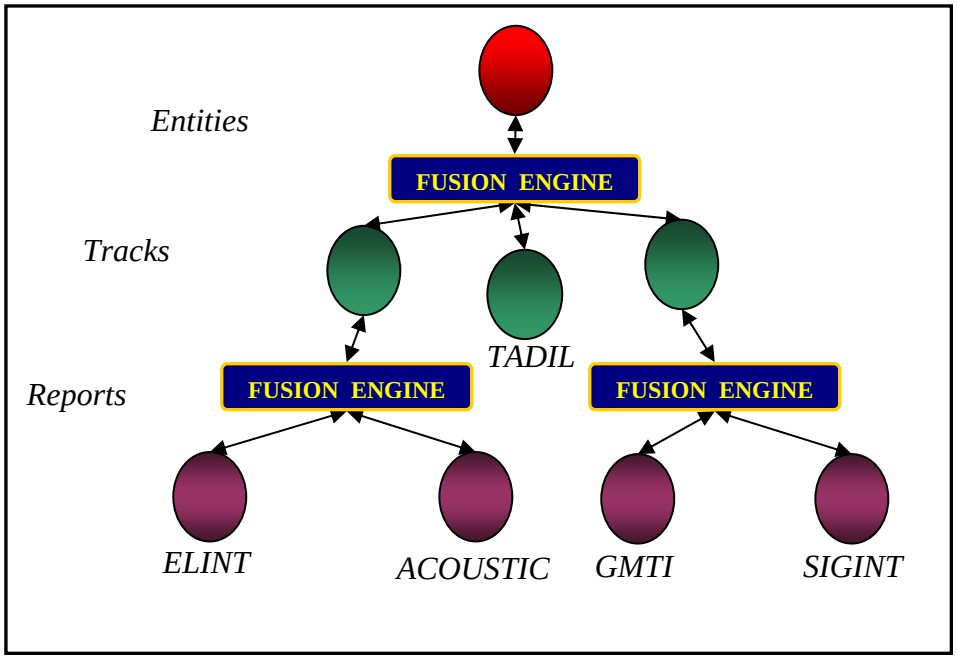

Figure 28. Trickle-up pattern applied to MDA

The trickle-up pattern provides essential partitioning for ontology definitions in support of service discovery and allows developers to build systems in support of the role the data source provides in a greater SOA.

Reports are the lowest most discrete form of content. They are a collection of facts that alone provide limited information from a single source. Reports in a C2 context include singleton human (HUMINT), signals (SIGINT), or electronic (ELINT) intelligence intercepts. For example, an ELINT report would contain details such as a frequency and line of bearing or for a SIGINT report could contain a voice pattern in the vicinity of a specific region. Reports are usually limited in scope and duration of

coverage and only begin to reveal higher order knowledge when they are aggregated with other reports or combined with data from other devices. A sample report shown in Figure 27, from a fielded system displays the type of data associated with Reports which include time, position, source of information, and level of certainty.

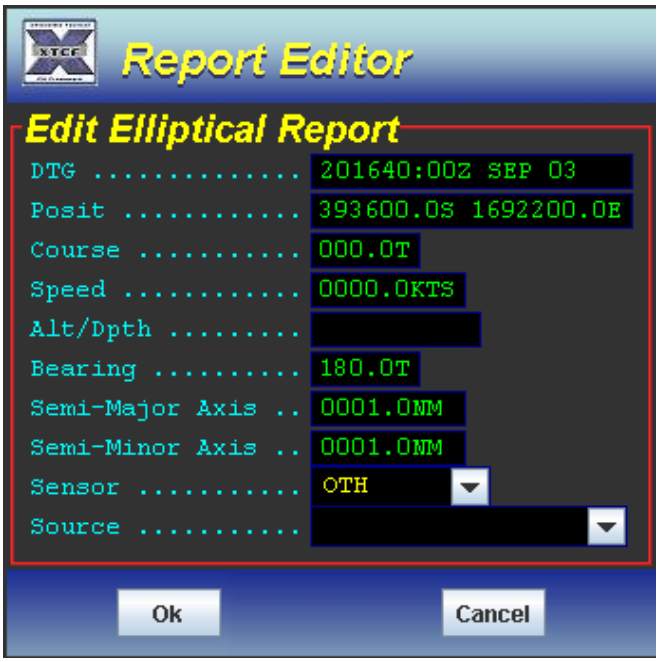

Figure 29. Sample Report Object

Reports provide the foundation of C2 system data. The next tier of data consists of tracks which provide a view of objects of a more actionable nature, built upon the facts provided by reports. Tracks are the objects normally associated with C2 systems and usually have kinematics associated with them. For example, a track may be comprised of a series of ELINT reports that over time reveal the movement and hence the course and speed of an object. Track objects are also formed from dissimilar reports by means of data fusion. For example, an ELINT report which reveals an initial position can be augmented by an acoustic report which further elaborates on the type and intended movement of the object and hence become a track. This dissimilar report aggregation and promotion to track relies on a fusion engine to evaluate the reports. There is a

capability to integrate at post runtime a fusion "service" that subscribes to a new report service to publish a track. Figure 28 shows some of the expanded data elements associated with Tracks.

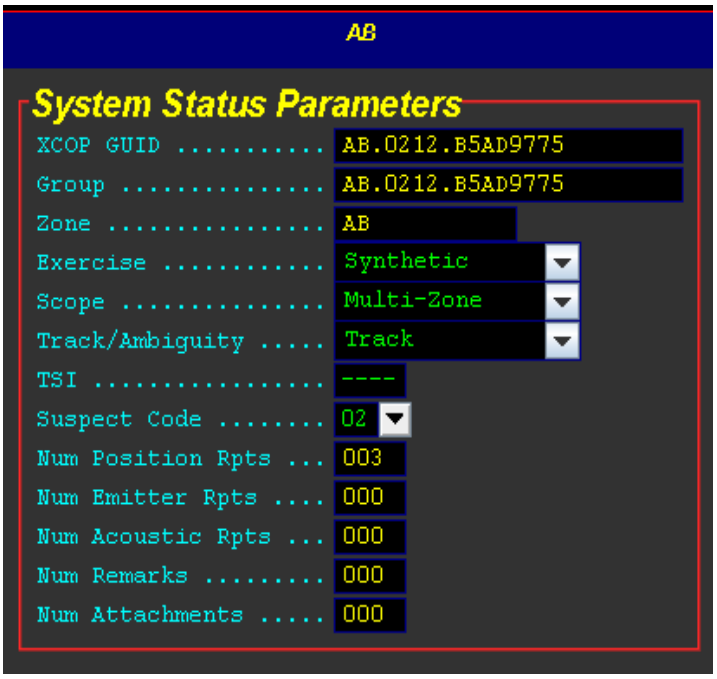

Figure 30. Sample Track Window

As discussed, tracks can be generated from a collection of similar reports. Beyond the kinetics (course and speed are not labeled in, this figure due to different approaches taken by the developers of the XCOP system, we have values such as Tactical Significance Indicator (TSI) which prioritizes the track update rate in support of asynchronous updating. Additionally note the Global Unique Identification (GUID) field, which uniquely identifies this object. By breaking the GUID into components we can determine from where the object originated and what system has ownership of the object.

The highest level of data object in this C2 paradigm is the Entity. Webster's dictionary defines an entity as "something that has separate and distinct existence and objective or conceptual reality." For this dissertation MDA discussion, entity refers to data objects with complexity beyond track or report level content. For example, we combine or fuse a track with a cargo manifest or a signal intercept. These objects

represent an information layer that exceeds the definition of tracks. Perhaps an Entity is a target folder, with routes, battle-space geometries/overlays and other associated objects identified along common lines.

V. SOA C2 APPLICATIONS-ZONE COMMON OPERATIONAL PICTURE TOPOLOGY

A. ZONE-DEFINED C2 PATTERN

Section one describes the emerging focus on Maritime Domain Awareness missions in stability operations, global war on terrorism (GWOT), and homeland defense. These missions drive a different type of Common Operations Picture (COP) topology than employed in major combat operations. In major combat such as naval carrier and expeditionary strike group (CSG/ESG) operations, the COP topologies employ traditional parent-child relationships in which all data is fed to the top-cop node at the top of the tree. The data is then replicated down the tree in an attempt to synchronize the data among all the nodes. The hierarchical *top-cop* topology supports unity of effort and a common view of the battle-space; however it wastes bandwidth and computing cycles as it moves data that not all nodes need. Additionally, a hierarchical COP has a diluted sense of purpose and mission as every warfare mission's content (e.g. air, surface, ASW) is mixed into a *top-cop* database. In contrast, the zone pattern creates a more flexible COP topology, which allows the commander to tailor the COP for a specific mission by only sharing content that is necessary for the mission. These concepts translate into software with two general requirements: data flexibility and ad-hoc partnering, these terms are discussed in greater detail below. Figure 31 illustrates the difference between the hierarchical and distributed COP topology.

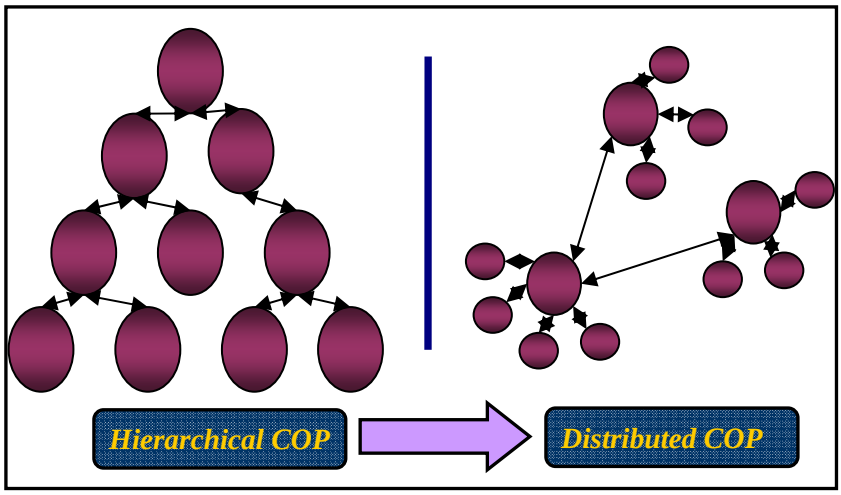

Figure 31. Hierarchical to Distributed COP

The distributed COP topology described above is enabled by building systems with two requirements of data flexibility and ad-hoc partnering. Data flexibility and ad-hoc partnering requirements are enhanced by using SOA technology. Data flexibility in the case of COP systems is the ease associated with adding and removing sources of content such as a new data feed or in the case of this research the ability to reach into the trickle-up pattern or reports, tracks or entities. Additionally, the new services that result from the auto data fusion process can be another source of data for the COP. Past COP systems were tightly coupled to the sources of data they managed. In SOA-based COP systems data flexibility is also defined as the ability to employ and manage a wide variety of data sources that employ standard data definition and transport technologies. A second component of the MDA desired COP topology is ad-hoc partnering, which is the concept of rapidly establishing, updating, and deleting COP data-sharing relationships. Three general roles exist in a COP which are similar to those in a tactical data link system. These roles are COP leader, participant, consumer, or silent listener (Logicon, 40). The following section discusses the interface between the Zone-COP model and the trickle-up pattern.

A SOA C2 Architecture has two main components: the infrastructure source components described by the trickle-up pattern and data management components described by the zone pattern distributed client model. The SOA Zone is the mechanism

by which a user/manager of the system interfaces with the data source infrastructure described in the trickle-up pattern. The Zone pattern created views from the content in the infrastructure layer. In much the same way a web browser utilizes the data from the Internet, the Zone model interacts with the trickle-up pattern. The Zone model manages the three relationships any C2 system has with its data and communications protocols. Figure 30 shows a view of multiple zones applied in a reference naval composite warfare command structure, where AB is the carrier strike group commander, AZ is the anti-submarine warfare commander, and AW is the air defense commander. Note the three relationships described the zone pattern, zone-to-client, zone-to-zone and zone-to-track or data feed.

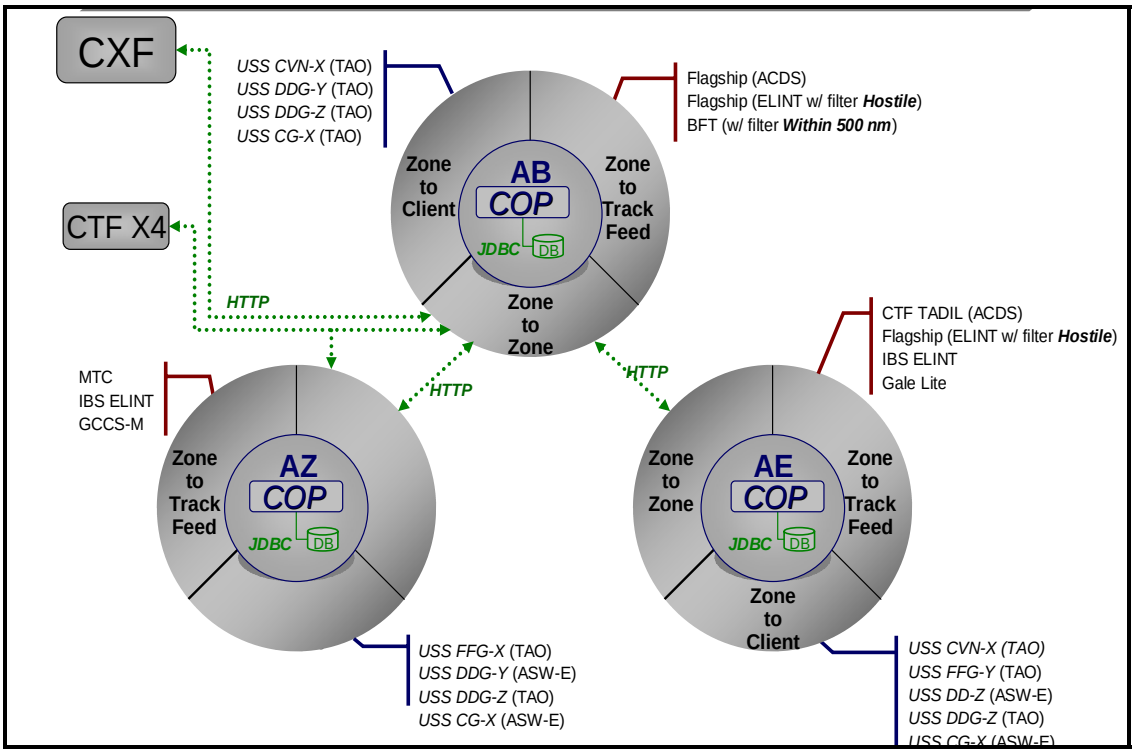

Figure 32. Zone Pattern applied to Composite Warfare Commanders

The Zone Pattern reflects the three intertwined services that the model must manage. Those three relationships are: the relationship between the user and the data sources the pattern requires, the relationship between the zone and other users who

participate with the owner of the zone in the validation and use of the system, and the relationship between the owner of the data and with other zones who consume the content. As described in the pattern description section, the third and second relationship parallels a magazine printer relationship with its fellow writers and readers, in that the content available to those who work at a magazine is different than the finished product provided to subscribers. In a C2 COP system, the raw or intermediate views that action officers are working on is different than the view data that they provided to a commander or senior decision maker.

The following central mechanisms enable the realization of the three relationships described in the *zone pattern:* visualization, collaboration, and data management. The visualization layer consumes content and displays it in a usable fashion. Some examples of visualization include map displays, tables, and graphs. A critical aspect for visualization is to apply SOA granular tenets and separate the visualization service from the data management, data source or other services, to ensure the content is not locked into one presentation mechanism. Figure 18 shows a Google display of Iraq, with reconstruction-site data displayed on it. Using eXtensible Mark-up Language (XML) transforms, the data was published from an SQL database to the Google Earth Keyhole Markup Language (KML) language. Attack data (position, type, time) data was also made available quickly for display by Google Earth, but is not shown here.

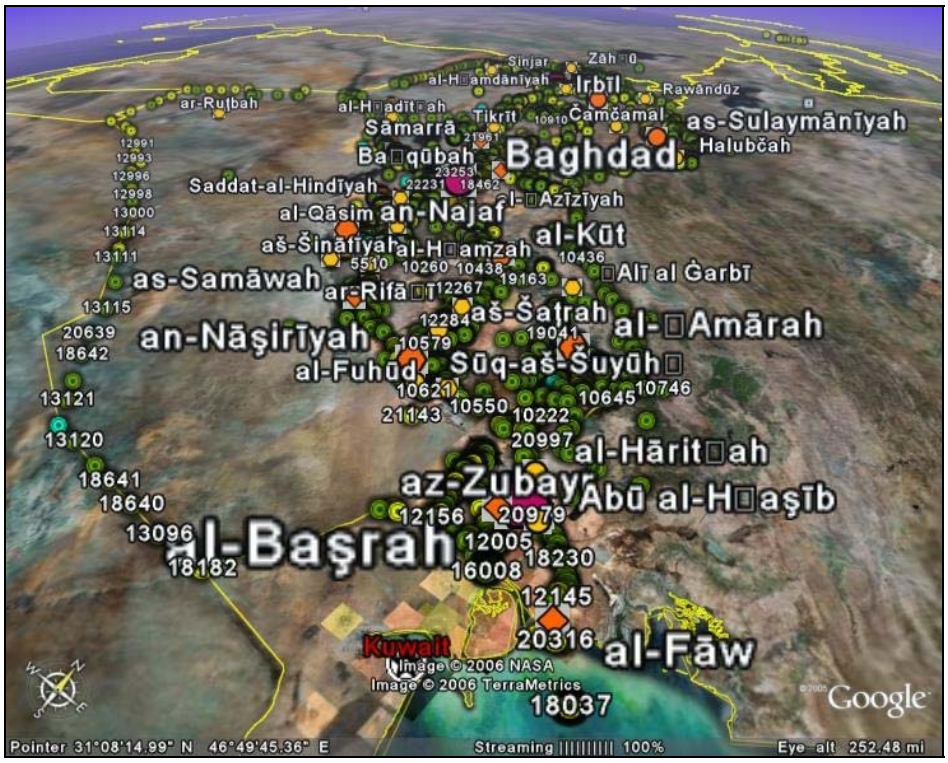

Figure 33. Google Earth Display of Iraq Reconstruction Projects

The next broad utilization of the Zone Model is collaboration. Collaboration is the sharing of data between users within the framework of the information system. In a C2 paradigm, collaboration can be facilitated by instant messaging systems and Voice over IP (VoIP) systems or collaborative white boards, among others. The essential component of collaboration design is not hard coding proprietary standards into the interfaces between the display systems, data management, and the collaboration software. Open standards such as eXtensible messaging posting protocol (XMPP) offer industry standards that allow systems to collaborate by passing XML messages with standard ports and interfaces, supporting SOA design principles. The results of these efforts are the ability to drag an object in display and share that object with another user who may be using different display software. The result is a C2 system with a seamless transfer of content between its collaboration and display system that is not locked into a specific vendor solution.

The final utilization of the Zone Model is data management. Data management provides a number of essential functions and executes much of the heavy lifting in the Zone model. The Zone Model is the main interface to the Core Services and Infrastructure. For example, it interfaces with the SOA discovery service to determine what is available for viewing, provides the persistence for the user views, and manages the data exchange between the manager and client systems. Let's examine more closely the role of the data manger in the Zone Model. Reviewing the roles of the Zone Model we have 1) zone-to-zone, zone-to-client, and zone-to-data-source data exchange. In each of these exchanges, it is the role of the data manager to maintain these relationships.

B. CROSS WALK RELATIONSHIP BETWEEN ZONE AND TRICKLE-UP PATTERN

In order to clarify how the trickle-up pattern *and* the Zone *C2* structure interact, let's trace the data elements through a reference implementation. The trace starts from a report sensor through various auto-fusion gates to an ultimate Entity, which is then consumed by a Zone Commander and then shared as a published COP. Although some implementation and supporting technologies are discussed to present the material coherently, greater detail is provided in later chapters.

Our hypothetical sensor is a remote acoustic sensor in the South China Sea. The acoustic sensor is a floating buoy, which transmits its reports via satellite IP connection to a Web Service Server located in Japan. The raw analog acoustic data is segmented into "Report Topics" parsed by an acoustic sensor into either probable source or classification of quality. These two *topics* are now *discrete* web-services that are categorized in a discovery index and can be consumed in follow-on tasking. A sample of an XML document of the content may include:

<Acoustic Report>move this to the next page

< Report GUID>XX-XXXXXXXXX</Report GUID>

<Report Classification> Possible Sub </Report Classification>

<Acoustic Frequency> 440Hz </Acoustic Frequency>

<Bearing> 120 </Bearing>

</Acoustic Report>

Further implementation of XML, messaging and discovery will be discussed later, however the take away from the passage is how granular the report may be, and how the complexity of who and how the data will be used later is hidden from the developer of a system at this level.

In order to demonstrate the utility of the data model, we will employ a second type of report. In this case we have an unmanned ELINT intercept site, again on a small atoll in the South China Sea. This device will record the presence and direction of radars or other emitters and place a report of that emitter on a topic in a similar fashion to the acoustic report discussed earlier. An example of one of those reports is as follows:

<ELINT Report>
< Report GUID>XX-XXXXXXXXX</Report GUID>
<Report Classification> Merchant </Report Classification>
<PRF > 440Hz </PRF>
<SemiMajorAxis > 1.0nm </ SemiMajorAxis >
<SemiMinorAxis > .5nm </ SemiMinorAxis >
<Bearing> 070 </Bearing>
</ELINT Report>

Two independent systems are automatically collecting dissimilar data from two systems. The persistence of each report is maintained by its own web-service without *a priori* knowledge of how the data will be consumed. The data can be accessed by a consumer via a type of web-service enabled SQL query *SELECT *FROM reports WHERE Classification = 'Merchant'* or as an event-based topic where each addition is broadcasted to whomever is subscribed.

In the example above, two report-level data objects are created by a sensor and maintained on a discrete web service operating independently to produce a web service. In the next phase we will examine how the content that, when fused or combined, is promoted to the *track* designation in the *three tier model*. In this phase of the reference model an Anti-Submarine Task Force Commander in the South China Sea Theater has established a fusion service that takes the inputs from the two report sources discussed

above to search within specified criteria for possible snorkeling submarine contacts. Due to the large amount of data, having an operator evaluate the data for matches would be too cumbersome. Instead the ASW Track Service "subscribes" to the output of the two track feeds and if it finds a correlation (a feasible object based on an acoustic contact correlated to an ELINT report with a PRF of a known submarine). This object is promoted to a track object by the fusion engine which is now available as a discrete web service. The power of the model is the orchestration of the services in a usable workflow, with dynamic bindings at run time, but more on that in later chapters.

In the next phase we have an intelligence analyst evaluating strategic intelligence across the battlespace. The specialist evaluates open source reporting and other intelligence reports that relates to a potential adversary's submarine exercises. The analyst is not sure who should view this data and decides to conduct a search of all the track sources for the area of interest. This analyst then creates an *entity service* that attaches the intelligence reports to all the track feeds in the area of interest. That entity data is now available as its own, discoverable, discrete, web-service for use consistent with the other levels of the tiers. Figure 32 shows the path the data in the reference model makes.

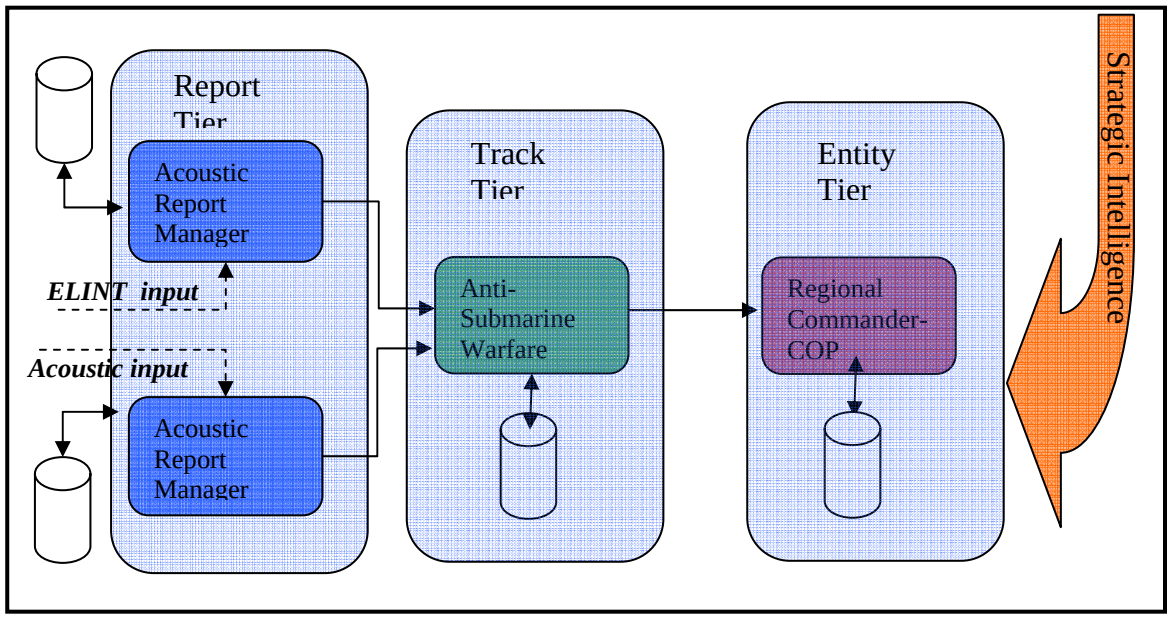

Figure 34. Trickle-up Pattern Applied

Presently our data has remained in the trickle-up *or* "service" side of the pattern. In this next phase, we will discuss how the data is utilized by the Zone C2 pattern. As discussed previously, the Zone Data manager component does a majority of the heavy lifting with respect to interfacing the content provided by the three tier data model and is the first touch point between the models. In the earlier reference model, we now have four data streams available for discovery: Acoustic Reports, ELINT reports, an ASW FUSION output and an Intelligence Analysts Entity stream. Looking on a UDDI driven search panel, made available via the data manager model, the set of available sources may look like the figure below, allowing the user to select which streams to subscribe to. Additionally, the sample UDDI diagram below utilizes a check-box interface to signify the ability to select more than one track feed. The ability to select multiple sources introduces complexity to the system such as data-ringing and duplicates if a data stream is selected which is comprised of duplicate primary sources. Mitigating methods such as global unique identifiers is a SOA challenges.

Figure 35. Sample UDDI Display

By selecting the track feeds above, the Zone data-manager now has determined the sources of data used to create the COP. We now have established one of the three relationships described in the Zone model. How the data is distributed and validated is the topic in the next section.

The remaining relationships discussed in the Zone pattern include *zone-to-client* and *zone-to-zone*. In the previous section, the content data has migrated from the limited report type through a fusion service and ultimately to an Entity Manager. At each stop of the tier, value-added content was added to improve awareness for its ultimate consumer. It is critical however to maintain the ability to obtain data from each level independently, to support other uses of the content. In the reference implementation under discussion, our Zone user has selected all of the data feeds available in the directory above. The user now has an initial COP, based on those sources. Thus, it is now time to add further value, share and collaborate on the view. The sources now have a local copy maintained in the user's system and that data is available for viewing via an open standards based XML/messaging interface. The operator now has an opportunity to make changes to the data, or perhaps execute user-based fusion and correlation activities. The commander now has two views of the battlespace, a raw view that is shared with staff, and subordinate commands that endeavor to improve the quality of the view and a "production" view that is available for decision makers and commanders. These separate views are in fact stored in the same location, but are filtered by query. The raw view is made available via SOA interface to the Zone to Client interface, and the production view is available via the zone-to-zone interface. The zone-to-zone interface is via a web service and can be thought of as an Entity Manager of sorts, with the exception that the content is from a Zone Manager.

The role of these views is ultimately communication: Communication to the user of the disposition of assets and adversaries, and the users communicating intentions. Creating or updating the view of the battlespace is not a mechanical endeavor, where data feeds are pored into a collective map and displayed. Battlespace views or COP are built over time by expert watchstanders, who tailor the data they receive to fit specific decision-making priorities. A critical, but often under evaluated aspect of the development of COP systems is the ability to rapidly communicate on that view. To communicate effectively on a view, the system must allow seamless interaction between components, to allow a user to "drag and drop" tracks in a chat room, or mouse-click on an object to highlight it on multiple users screens. Accomplishing the communications, a

software architect may be tempted to create a monolithic system; however this path is inconsistent with the SOA design tenant, and can be avoided by utilizing common standards and interfaces.

In this section we will stand up a fictitious C2 structure utilizing the Zone and trickle-up pattern established previously. A Navy Carrier Expeditionary strike group is operating in the South China Sea. Embarked an Aircraft Carrier is the Sea Combat Commander (SCC). The SCC establishes a zone and populates it with content from the three tier model. The Zone Manager then establishes its zone-to-client relationship with its subordinate watchstanders and other platforms that assist the SCC with building the COP. These zone-to-client relationships are analogous to the host/client relationships in an Internet chat room, and allow collaboration on the battle-space view provided as a raw view. Next, the Zone Commander will make its "product view" available to superior or peer commanders, such as the Air Defense, Group or Strike Commander, for use in their respective battlespace views. Moving beyond the basic concepts behind the zone and three tier models, we will now discuss how the nature of COP and traditional systems differ.

C. CONTRASTING TRADITIONAL C2 STRUCTURES TO DISTRIBUTED ZONES

A number of technical influences and limitations from both the commercial and military sector have influenced the development of COP systems. Early successful client-server systems such as Automated Teller Machines and the Tactical Data Link system still in use today were guideposts to developers building PC-based C2 Systems. Section two of this paper provides the reader with background information about these systems and how they influenced COP systems. In this section, we contrast software architectures of past COP systems with those of SOA-based systems.

One of the central tenants of earlier COP systems is a "common" view. The common view is facilitated by synchronizing a series of databases though out the network to a master view. In Naval C2 systems, the master database is usually located at the command center. Tracing the path data follows a traditional hub-spoke architecture, in which subordinate commands report new objects to a central database, which then

merges the objects and sends a common picture back out to the subordinate commands. This type of architecture is employed by both the Navy Tactical Data Link (TADIL) and Global Command and Control System Maritime (GCCS-M). In traditional Navy Strike Group roles, an Aircraft Carrier of other Command platform would host the master database, and escort ships, submarines and air platforms would report. Presently, limitations associated with the latest C2 requirement is for a more distributed and dynamic structure, stretching systems originally designed for static reporting environments. The most significant difference between older systems described above and "net-centric" systems is who and how the system is configured. The analogy is similar to differences between mainframe client-server systems and Internet-enabled personal computers. In the case of GCCS-M, the master database acts more as mainframe and the supporting commands play the role of the clients. In reality, the GCCS-M software allows assignment of any unit as the master although it is the complexity of configuring the system and the forcing function of its requirement to establish a command hierarchy that limits its ability to allow more dynamic assignment of COP responsibility. In a net-centric system, every system can easily shift from a supporting to a command role and quickly establish the relationships between its data sources, consumers and clients. The techniques employed to synchronize the data are similar in both older and newer systems, and hence at many levels, the methods are identical. It is how the complexity is managed and hidden to the user vice the low-level methods that differentiate the older and newer methods.

Although both the Parent Child and Zone methods are similar at a macro level in employment, a discussion of specific implicit requirements derived from facilitating the ad hoc dynamism discussed above will reveal further differences. As discussed previously, the state of the art in memory, bandwidth and software complexity restrained COP system design in a similar manner to any other technology. Those restrictions manifested themselves in hardware/software solutions tailored specifically for the host platform (such as an avionics platform or shipboard combat system) or rigid in interface, making modifications or changes difficult. In addition, the rigidity of interfaces limited the ability of early COP systems to interface with other systems and provide their data to

other consumers. Without the ability to easily add new types of data objects and sensors, the system architecture would become unwieldy. Additionally, when forced to interface, the complexity was high and interfaces were brittle and difficult to manage. Another manifestation of the technical limitations was an inability to rapidly incorporate and configure modules to properly correlate, fuse and safeguard data (to include emerging cross domain data exchange), limiting the ability of the system to respond to emerging requirements.

Beyond the difficulties of interfacing, the state of the technology also limited the methods used to store, transport and manage data. Early systems had significant memory and bandwidth limitations. However these limitations abated over time, but the underlying software architecture did not alter its design to the emerging reality. Data distribution remained "dumb pushes" where whole databases where synchronized vice a system where only the items required (geographically) where moved, wasting bandwidth and CPU cycles. Additionally data ownership and pedigree was limited, allowing users to perform delete and modify operations they should not have access to.

The Internet model is the basis of the Trickle-up and Zone patterns and offers many solutions to the above limitations for future Common Operations Picture Architectures. These enablers include dividing the architecture into multiple layers and discrete services vice the API method employed today. The broad layers include a data source layer, a data management layer and a consumer layer (such as display). Forcing the architecture to utilize a common messaging and standard data schema such as XML between these layers starts to develop the Service Oriented Architecture framework allowing extensible additions of new data sources, consumers and "business to business" (B2B) applications on the architecture. This will move the system from a "track-centric" display system to a fully integrated Intelligence Framework with numerous uses and consumers to leverage and compose the desired data across the enterprise. To function, SOA's rely on discovery services, common messaging and smart pushes and pulls based on user requirements. SOAs return the ability to tailor data to meet specific mission needs in a just-in-time format in a standards-based, hopefully cost-effective framework.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. MULTI-SOURCE DATA FUSION DISCOVERY SERVICE-AUTO-FUSION

As discussed in section one, web-services or WS* standards provide a foundation computing environment to build loosely coupled systems. Utilizing these the research aims to create an auto-fusion process to execute the following steps to join disparate data types.

- Service is selected for Auto-Fusion using BPEL
- Attributes are compared by a discovery method for valid services candidates
- Based on criteria, Fusion Engine is selected and data fusion is executed
- New Services are created by BPEL
- User validates if new service should remain

Figure thirty six below diagrams the auto fusion process. On the left side are the SOA capabilities; discovery, orchestration and services. These capabilities map to the following web-service standards; UDDI, BPEL and Service respectably.

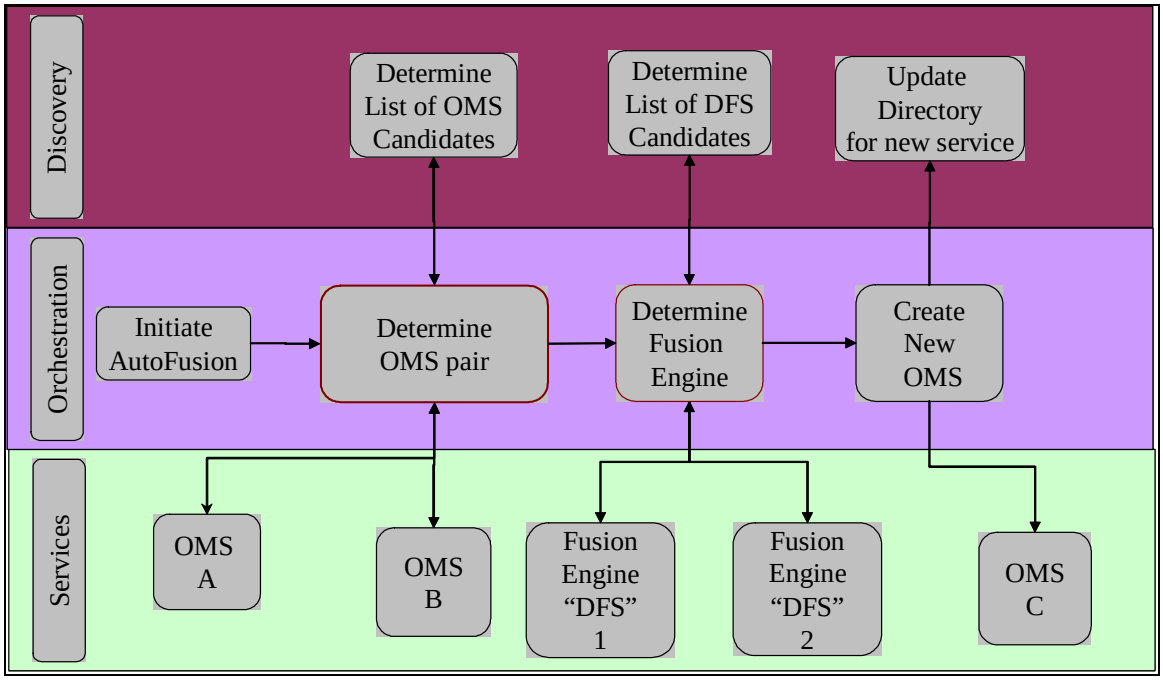

Figure 36. Auto Fusion Process

As discussed in section one, the *auto-fusion* process provides a broad requirement that drives the data strategy for an entire control and monitoring fusion framework and the *trickle-up* and *zone* patterns provide the foundation technology constructs to enable auto-fusion. Assuming we have a *trickle-up* foundation of sensor data and fusion algorithms, the next issue is how do we rank and match them to support improved awareness of the environment; the *auto-fusion process* starts to address those challenges. Figure 34, shows a process, but the central question is criteria of data source and fusion logic matching. Section two discusses the joint directors of labs level of fusions and provides a basic construct for addressing the challenge. The JDL levels of one through four give a rough framework of the types of fusion, but do not necessarily provide a ranking, such as level four is better than level one. It does however provide thoughts on the complexity of the fusion.

From a software engineering perspective the potential cost of a fusion operation is an interesting metric to examine for evaluating fusion participants. Less qualitative a metric than "potential improvement in situational awareness," cost potentially could provide a method to evaluate fusion participants. Following a cost model, propose an *Auto-fusion figure of merit* comprised of three cost related metrics computational complexity, network cost and a JDL target cost.

Auto-Fusion FOM = Computational Cost + Network Cost + JDL Target

The computational cost is an evaluation of how complex the algorithm is, for example, in the fusion domain an attribute matching algorithm may be less complex than a position estimation algorithm. For network cost, a view of how many network hops are required to reach the data source of fusion algorithm. Network cost could also factor in the existing bandwidth available. JDL target reflects the opportunity cost of a failed attempt to fuse. The concept is, if we attempt to fuse objects that are unrelated, or attempt to fuse with an unsuitable algorithm we have lost time. The JDL target reflects that if we are fusing two reports or two tracks level objects they are more likely to fuse successfully.

For the Auto-Fusion FOM to work in a SOA framework where services are paired with fusion algorithms post run-time, the services must include metadata to tell a discovery service where it is for network cost, how complex it is for computational cost and what type of data service it is to influence the JDL target (report, track or entity). The motivation for using the trickle-up pattern is the ability to place the services in categories to aid in estimating the JDL target and hence the FOM.

The prototype software developed in this research addressed some early methods to supporting the auto-fusion process. The prototype system section goes into greater detail on how a system could approach these challenges and provides reference architecture for potential implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. MARITIME DOMAIN AWARENESS (MDA) SYSTEM OF SYSTEM CHALLENGE

The US *National Plan to Achieve Maritime Domain Awareness* defines MDA as "the effective understanding of anything associated with the global maritime domain that could impact the security, safety, economy, or environment of the United States." MDA is a Joint DoD, Department of Homeland Defense (DHS), partner nation and intelligence services effort to use a defense-in-depth strategy to support the following United States goals for MDA (MDA Strategy, 1).

- Enhance transparency in the maritime domain to detect, deter and defeat threats as early and distant from U.S. interests as possible
- Enable accurate, dynamic, and confident decisions and responses to the full spectrum of maritime threats.
- Sustain the full application of the law to ensure freedom of navigation and the efficient flow of commerce.

US Navy, DHS and US Coast Guard stakeholders have defined MDA into four pillars of MDA information; vessels, cargo, infrastructure, and people (CONOPS, 2) . For each of those pillars two broad activities are defined; Global Maritime Intelligence (GMI) and Global Maritime Situational Awareness (GMSA). In the MDA CONOPS, the following formula is proposed:

MDA = GMI + GMSA

GMI are the intelligence activities in the maritime domain to determine threats, while GMSA refers to the persistent monitoring of the maritime domain. The CONOPS defines GMSA as "comprehensive fusion of data from every agency and by every nation with knowledge of the maritime domain." For each one of the MDA pillars (vessels, cargo, people, and infrastructure) the CONOPS proposes an enterprise hub for each that coordinates the GMI and GMSA activities. Various agencies would lead the enterprise hubs, with a goal of creating a multi-level security, persistent user defined operational

picture (UDOP) of the maritime domain. Figure 5 below, from the MDA CONOPS details the multiple agencies and enterprise hub concept employed by MDA.

Figure 37. MDA Enterprise Hub Architecture (MDA CONOPS)

A number of technical challenges exist to transform the MDA requirements into a tangible system. In the next section we will discuss the broad technical approach adopted by MDA, and how the research in this dissertation can address some of those challenges.

A. MDA TECHNICAL CHALLENGE

A number of technical challenges exist to create the complex system of systems envisioned by the authors of the MDA CONOPS. A number of them are addresses in the document itself to include; numerous isolated databases of information, large expanses of ocean areas are unmonitored, incompatible data exchange methods and extensive policy and cultural procedures inhibiting information sharing (CONOPS, 4). At a macro-level, MDA has adopted web-enabled SOA as a means to integrate numerous systems and address interoperability challenges. These interoperability challenges include efforts to

integrate information from various commercial, multi-national coalition, DoD, intelligence agency, and non-governmental organizational data of various pedigrees and legacy infrastructures.

Although a web-based SOA provides a compelling overall information architecture technical solution, SOA alone does not provide all the technical tools needed to address the challenges. To employ SOA, MDA requires a number of supporting policies such as an overarching data strategy, which includes an extensible data model and governance structures. A number of efforts are underway across the DoD to provide the needed processes and systems to develop interoperable systems, to include MDA related XML schemas and data strategies. One key challenge, discussed specifically in the CONOPS is central to the dissertation; the significant data management challenge from MDA. Specifically, the CONOPS cites, *"the expected day-to-day MDA operations will generate incredibly large amounts of data and information."* Additionally the CONOPS discusses a method to address the large amounts of data expected to result from MDA operations, the CONOPS directs the development of an "*automated fusion capabilities that integrate into net-centric architecture."* These two concepts are central to the software patterns, models and methods in this dissertation.

Creating an auto-fusion architecture envisioned in the CONOPS is not a simple appendage or requirement on a broader system, but a core influence on the entire architecture of the solution. Although little detail is provided in the CONOPS, an automated fusion capability in the maritime domain would take an identified source of data or single object and attempt to find suitable data source and fusion algorithm to combine it with. The MDA CONOPS refers to data fusion as "combining of automatically correlated information with the data that refines the information or presents it in intuitive format."(CONOPS, A-8). This definition aligns with the definition provided by Steinberg and Bowman which states "fusion is a process of dealing with the association, correlation and combination of data and information from single and multiple sources to achieve refined position and identity estimates, and complete and timely assessments of situations and threats, and their significance"(Steinberg, 2-3). The MDA definition branches out into presentation and seems to view correlation as a

predominantly machine activity, vice a human analysis activity. Although I can find no specific studies on the ratio of human processing to automatic, limiting the discussion of fusion to automatically generated content would ignore a range of content that only informed operators can provide, and is not aligned with the Zone Pattern discussed in the dissertation. Utilizing the definition provided by MDA and augmenting it with Stienberg's we can begin to realize the scope of an auto-fusion framework and address how this requirement drives the data strategy and architecture for the entire MDA system. The patterns detailed in this research have direct applicability on the MDA auto-fusion challenge and are detailed in the proceeding sections.

B. MDA EMPLOYMENT OF THE TRICKLE-UP PATTERN

The Global Maritime Situational Awareness (GMSA) detailed by the MDA CONOPS requires a varied and significant number of sensors geographically dispersed throughout the world. The *vessel* pillar is especially "sensor" driven and is a classic tracking problem on a global scale. A number of sensors can track the movement of a vessel trough the water. These include ELINT, SIGINT, acoustic, radar, automated information system (AIS) and national sensors. Each sensor has its benefits and drawbacks. ELINT or electrical emissions are normally tracked from target vessels radar set. ELINT accuracy can be limited, but has long ranges and under certain circumstances can provide identification attributes. SIGINT usually results from communications gear aboard a target vessel and has similar limitations and identification attributes as ELINT. Acoustic sensors have tremendous ranges, but often lack the identification and positional accuracy needed for a complete understanding of the vessel. Radar is the traditional sensor for position; however it lacks identification and has limited range. AIS is a system of shipboard transceivers which the International Maritime Organization's (IMO) Maritime Safety Committee directed certain ships over 300 gross tonnage carry. The system provides information on the identification or Maritime Mobile Service Identities (MMSI), position, course and speed of vessels and is rapidly becoming a critical part of the vessel tracking challenge. AIS is an excellent source however it is often in complete and those engaged in illicit behavior are expected to employ AIS in a deceptive manner, limiting its effectiveness, Additionally, AIS vessels may unintentionally reduce the

effectiveness with poor antenna placement of improperly configure the device. Lastly, although national intelligence agency content is a critical component of the MDA solution, it is beyond the classification of this research, however no apparent reason exists that the patterns and methods discussed are applicable to that content as well.

Executing the global tracking required of the GMSA, all of the above sensors are required to build as complete a picture as possible. Installing remote sensors in vicinity of strategic sea lines of communications will create a global grid of content, which needs to be organized, maintained and modified to suit the tactical environment. The MDA framework is an ideal candidate to employ the *trickle-up* pattern. Specifically, the *trickle-up* pattern describes a basic ontology for the remote sensor data as report, track or entity. Furthermore, the pattern articulates how the fusion and correlation algorithms operate independently to the data sources until runtime and each data source and fusion algorithm are unique SOA services. This agility allows the incorporation of new sources and locations into the data framework as well as allowing different algorithms to operate on the sources. For the global, geographically dispersed MDA requirement the pattern provides essential scalability and maintainability. A full description of the trickle-up pattern is provided in the dissertation, along with a prototype system as reference architecture for the MDA pilot.

C. MDA EMPLOYMENT OF THE ZONE PATTERN

As discussed previously, the MDA CONOPS envisions a globally netted group of remote sensors providing situational awareness over the maritime domain. This notion however negates the significant value human operators can play in evaluating sensor data and the traditional means of building a common picture of the area. The *Zone Pattern* provides a method for operators to manage the three relationships a monitoring system has; one, the relationship it has with its automated sources of data such as those provided in the *trickle-up* pattern, secondly, the relationship a zone has with its subordinates collaborating on the contents and thirdly the relationship a zone has with the consumer of its authoritative content. We examine those relationships as they apply to MDA in the following pages.

A key aspect of the *Zone Pattern* is the concept of accountability for the quality of the view of the battlespace. One of the challenges discussed earlier is the view of some Command and Control system developers that situational awareness is achieved simply by exposing data sources in a SOA construct, and allowing those sources to be subscribed by a visualization application. The MDA CONOPS architecture is defined by the user defined operational picture (UDOP) constructs, which articulates the above view. UDOP architecture envisions a source to user view that fails to recognize the complexity of merging sources, methods and fusion systems to build a picture that decision makers can fight from. The ***zone to zone*** relationship captures this concept that sources and collaborators work on a common view and prepare a finished view for those who subscribe to the authoritative view. In figure six below a view of the Strait of Malacca between Singapore and Malaysia is shown. The area is a pivotal sea lane of communication (SLOC) and a significant portion of world energy and commerce supplies transit the waterway. Any MDA strategy would target the area as a good choke point to position numerous sensors to track vessels transiting the waterway. Two nations border the waterway each with different relationships to the U.S. Following the Zone Pattern a series of the sensors would feed a Zone which in turn would correlate the various sources and provide the content to consumers. The U.S. may be one consumer, and the southern nation may have certain filters set to only provide information stipulated in a data sharing agreement with the US (perhaps the nation may filter out its own military units, ect). The northern nation may have different data sharing agreements with the US and the southern nation and provide different content to each.

Figure 38. MDA Sample Sea Lines of Communications View

The next relationship in the *Zone Pattern* is the ***Zone to Client***. Here the pattern articulates a different relationship between what a Zone shares with its consumers and those who collaborate on it. The *Zone to Client* construct is a "raw view" as operators work to collaborate on the objects managed by the view. Perhaps on this view, uncorrelated ELINT hits and AIS reports with no identification are worked on by operators to improve situational awareness. The above type of low quality data would clutter the decision maker view and is usually not actionable enough for decision makers. In an MDA system a number of participants may participate in a Zone, perhaps by region or mission. For example, in the Strait of Malacca area discussed the strait may be divided into sections with a Zone assigned to each. In the Zone perhaps one mid-size vessel, two shore station operators and a regional headquarters as the "Zone Commander" are joined in a Zone. Among the participants in this zone, a raw unprocessed view is created and collaborated on. The Zone leader may sit at the regional headquarters, decide what is actionable and promotes those objects to the Zone to Zone or "national view."

The last relationship is the ***Zone to Data Source.*** This relationship is between the Zone and the data sources it subscribes too. Data sources are automatically generated

collect from sensors like AIS, ELINT or RADAR. These sources are part of the *trickle-up* pattern and key component of the Zone to Client relationship is the concept that a Zone could reach into any level of the trickle-up pattern of reports, tracks and entities. Expanding on the MDA Strait of Malacca example where regions were divided into Zones. In this example a Zone Commander selects a number of remote data sources to subscribe too. These may be a radar station, ELINT and AIS device within the area of responsibility, as well as the AIS receiver on the ship. These sources now populate the common view of the Zone.

The MDA challenge is significant in both its scope and geographically dispersed nature. The *Zone Pattern*s cellular COP strategy provides the robust, scalable pattern to manage the raw and operator value added content. Enforcing accountability and decision making at the echelon of command suitable for the data analyzed the pattern departs from both the TOP-COP model where every node is a mirror of the commanders and the UDOP model where every source is fed into one general repository where users subscribe to the desired content, but do not add value. The *Zone Pattern* reflects a realistic view of data sources, clients and partnerships where operators "create" a view of the battlespace from sources and subordinates that is cleaned and analyzed prior to making it available decision makers. In the next section we will discuss how an Auto-Fusion process can employ the *trickle-up pattern* to provide higher value content to the *Zone Pattern*.

D. MDA EMPLOYMENT OF THE AUTO-FUSION METHOD

The MDA CONOPS recognizes the vast quantities of data an MDA system would create from not only from installing new sensors on various waterways, but by joining existing sensors to a broader grid. The MDA information architecture needs to address those challenges and the *trickle-up* and *Zone* patterns provide a foundation to address them. An additional approach endorsed by the MDA CONOPS is an auto-fusion process discussed earlier. The auto-fusion process would allow an MDA analyst to dynamically discover and orchestrate a group of services to search for trends and anomalies. In the projected MDA information architecture, with dynamic addition of data sources, fusion engines and participants a dynamic discovery method is essential for analysts to employ the most current sources and tools. The auto-fusion process provides that high level

requirement, which in turn provides an implicit requirement that drives the entire information architectures structure and strategy.

For example, in the previously discussed potential MDA subsystem in the Straits of Malacca, an MDA analyst may be sitting in a regional headquarters. The analyst participates in one of the regional zones and is charged with maintaining as good as possible view of vessels transiting through his or her area of operations as possible. The analyst has a AIS remote feed and decides to use the *auto-fusion process* to determine if a better source or fusion algorithm is available to analyze the content. Presently the analyst has an AIS report, however the unit on the target vessel is malfunctioning or misconfigured and providing no identity information. Starting with the initial report, the analyst activates the auto-fusion process which takes the data and meta-data on the original report and searches for suitable pairs. First the discovery service determines that a service with similar attribute based information is available in the region of the operator and determines a geofeasability based fusion engine is the best fit to create the next level object. The end result of the auto-fusion process is pairing the AIS reports with ELINT reports just joining the domain since the ELINT LOB reports stem from a transiting coalition ship. The two are fused using a geofesability fusion engine since the AIS has no attributes and the ELINT provides some classification (merchant) and a wide ellipse. The example shows how dynamic the analyst's environment can be. Although the AIS transponder may be a constraint in the analyst's work flow, the ELINT from a passing ship may not. Additionally, as new fusion and correlation engines become available they can be added to the environment dynamically at runtime without significant changes to the existing system. The auto-fusion process has implications on a wide Varity of data strategy and information architecture issues which are discussed in the next section.

E. MDA DATA STRATEGY

On May 9th, 2003, John P. Stenbit, Chief Information officer for the DoD released a joint service *data strategy* to support the defense departments Net-Centric goals. The strategy states that "the core of the net-centric environment is the data that enables effective decisions" and "data implies all data assets such as system files, databases, documents, official electronic records, images, audio files, web sites, and data access

services" (DoD CIO, 3). The data strategy articulated a number of goals and milestones to support the DoD net-centric vision. These goals stipulate that data should be; visible, accessible, institutionalized, understandable, trusted, interoperable, and responsive to user needs. These broad goals also support the DoD's maritime domain awareness community of interest as well.

The MDA COI is a large and diverse group and a unifying data strategy is critical to interoperability. The MDA data strategy should include standard artifacts to include, service level agreements, data definitions, lexicons, ontology's and a data schema for each of the four pillars (vessels, people, infrastructure and cargo). Gaines and Michael's define service level agreements (SLA) as "agreements that describe requirements and incentives for meeting performance thresholds, and specify incentives for meeting performance and QoS requirements" (Gaines, 289). One of the essential elements of a data strategy are the data schemas that describe the data objects. A great deal of effort and research on data schemas and definition are available to MDA developers. If systems have transitioned to an XML defined object, the present definitions are often well thought out and ready for adoption and for systems not transitioned to XML, the legacy data formats are good sources to start and provide insights into how the objects are employed. Conducting the research for this project, a number of DoD entities are involved in the development of definitions and schemas. For MDA, a number of entities are involved in development of data definitions and schemas, providing guidance to MDA developers and significant standardization has been made by other communities of interest. For example, for the "people" pillar, the intelligence agencies data definitions provide a good foundation, for "infrastructure" and "cargo" the commercial shipping industry has good definitions and for "vessel" the navy and maritime intelligence services provide a common reference. The key issue is not starting from scratch and learning from what exists and is adopted. As with any schema or definition, some elements will remain unique to the MDA COI, however this should be the exception vice the rule. The focus of this research is associated primarily with the "vessel" pillar and will focus next on how the two patterns and auto-fusion process can positively influence the MDA data strategy.

1. MDA *Trickle-up Pattern* Data Strategy Mapping

Providing global situational awareness of worldwide maritime vessels traffic is a significant challenge. A data strategy Developing a data strategy to support the MDA vessel pillar requires recognition of the dynamic nature of the environment. Sensors of various pedigrees will join and leave the "grid" at various times at the direction of fluid coalitions and general equipment availability. Supporting the dynamic quality of the MDA domain, the *trickle-up pattern* articulates the relationships and activities of sensors and services. As discussed previously, each sensor is a unique contributor to the MDA framework and as each sensor joins the framework it can be dynamically paired with other data sources and fusion algorithms. The prototype software developed for this research employed a business process execution language (BPEL) engine to join and invoke the services. Applying lessons from the prototype and the *trickle-up* pattern, the MDA data strategy needs to provide or reference the following artifacts:

- Common XML schema for MDA vessel type (Track in the t*rickle-up* pattern)
- Common XML schema for each sensor type (Reports in the t*rickle-up* pattern)
- Common service level agreements (SLA's) for each sensor or report type:
    - Sensor availability
    - Sensor expected performance
    - Sensor expected uses and partners

The artifacts provided by the data strategy discussed above provides guidance to developers on how to build services that reside on the MDA framework and users on what expect from a sensor. Next we will discuss how the *Zone Pattern* in the MDA framework supports the interoperability goals and makes the content available from the "report, tracks and entities" level to the Command and Control level.

2. MDA *Zone Pattern* Data Strategy Mapping

The *Zone Pattern* provides guidance to MDA developers building the applications that receive the inputs from the sensors feeds and maintain the objects they create to visualize, collaborate and manage the view of the target maritime environment. Similar to the *trickle-up* pattern influences on the MDA data strategy, the *Zone Pattern* guides the

interactions between sources, operators and consumers, and provides input to service level agreements between providers and consumers as well as input to the definitions of a vessel object. As discussed previously, the *zone pattern* articulates the relationship between an instance of MDA authority and the data sources, participants who assist in the development of the view and those that consume the "finished" view. Specific inputs to the MDA data strategy include:

- Service Level Agreements between data provider and application
- Service Level Agreements between "approved view" and consumer
- Data definition and schemas related to vessel objects

For example, in an MDA system, a report level sensor such as an AIS receiver would feed a report manager in the *trickle-up* framework. The report would be fused with reports from an ELINT source and promoted to a *track* level object. This *track* level object would then be subscribed to by a *zone*. These zone participants would work to resolve ambiguities and create an authorized view of the defined region or mission space. The data strategy would provide guidance to developers of the AIS report manager to what schema and data definitions to build to as well as how to interact with the fusion engine logic existing in the framework. Additionally, the data strategy would provide artifacts describing how the *track* level managers would interface with the Zone applications and provide minimum pedigree standards for the data such as how often the objects are updated and how long of a history the data store maintains. Referring back to our example a statement such as; "to participate in the MDA framework an AIS report manager shall maintain a report history of twelve months, with an update rate of 5 minuets, have a general availability of 95 percent and have demonstrated interoperability with MDA fusion engines" A similar statement could apply to *track* or *entity* level objects, with defined transforms between the *trickle-up* and *zone* managers. The Zone may have a data strategy statement like "MDA Zones limit high interest objects to 10% of expected daily track load to avoid congestion." We will close the discussion of with a discussion of how the auto-fusion process should influence the MDA data strategy

3. MDA *Auto-fusion Process to* Data Strategy Mapping

In Robert Persig's book *Lila*, he divides the vague world of "quality" into two broad categories of static and dynamic quality (Persig, 172). An interesting parallel exists between Persig's view of patterns and the relationship the *auto-fusion process* has with the *trickle-up* and z*one* patterns. The *trickle-up* pattern is intended to represent a more static pattern (although relatively agile when compared to traditional software); while the *zone* is a much more dynamic pattern representing the minute by minute changes and value added activities. The *auto-fusion* process provides a unifying intellectual framework to unite the two patterns in a real world example and provides the mechanism to translate the dynamic changes in the environment to solid static patterns that a system can employ.

Translating the concepts above to the MDA paradigm, the *Auto-Fusion process* provides the MDA data strategy with a unifying goal and an implicit requirement that enforces interoperability. Where the *zone* and *trickle-up* patterns divide the MDA vessel pillars into data sources (e.g. static) and consumers (e.g. dynamic), the *auto-fusion* process allows the architecture to renew itself as participants grow and chance. For example, referencing back to the AIS report example cited above, we enable the *auto-fusion process* to look through its listing of other data sources and fusion engines to join with. To facilitate a framework that accomplishes those goals the data strategy must detail and enforce a number of guidance measures. Specific guidance statements include;

- How data sources and fusion engines described in the discovery service.
- How fusion engines are ranked against each other based on applicability towards existing and projected fusion pairs.

Mapping the requirements of an auto-fusion construct to the Maritime Domain Awareness system is a high order requirement that drives a number of lower level requirements driving interoperability between a number of sources and systems. The MDA data strategy is the vehicle to inform developers of the nature of the framework and influence the architecture to support the desired interoperability and functionality.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. PROTOTYPE SOFTWARE SYSTEM

A. INTRODUCTION

The purpose of the prototype software system is to demonstrate the technical feasibility of a SOA web service implemented *three tier data model* to the Intelligence, Surveillance and Reconnaissance military community and validate the potential knowledge management benefits associated with its adoption. The prototype software development occurred in two phases. Initially the prototype was a traditional application explained later in this chapter. The traditional application was developed by the author to provide a geospatial representation of attack and reconstruction data in support of coalition objectives during Operation Iraq Freedom (OIF).

In support of this dissertation research, the application was adopted to a SOA based collection of web-services. The programming logic of the traditional application was segmented into a number of web-service components and chained using a web based orchestration module. This chapter will provide detailed design description and requirements starting first with the legacy application, followed by the SOA web-service version. The chapter concludes with a discussion of how the SOA based version validates the three tier data model and how covers some of the technical challenges associated with developing and implementing the prototype to include recommendations on changes to some web-standards.

B. LEGACY SYSTEM DESIGN

The legacy system was named Reconstruction Analyst Toolset/Common Operations Picture was a rapid development project chartered by the Security Plans and Operations directorate of the Iraq Gulf Region Division to facilitate analysis of Iraq reconstruction and attack data in a Geo-spatial format. The software takes two independent sources of data and conducts a simple geo-feasibility test similar to the some computations required in basic positional data fusion. Although the requirement at design did not drive a web-service solution, transition of the legacy system to a web-service provides insight into how the three tier data model could be architect using web-

service technology. RATCOP was originally developed by the author in April 2006 due to limited commercial software to analyze reconstruction data, and display it on mapping software in Iraq. Figure 4 below is a screen capture of the graphical user interface for the Reconstruction Analyst Toolset. The system allows the user to select search criteria on the GUI to search a database of insurgent attacks and reconstruction projects. The results of the search are written to a Keyhole Markup Language file which Google Earth can read to plot on its interactive map or a Comma Separated Value (.CVS) file that the falcon view map can read and display.

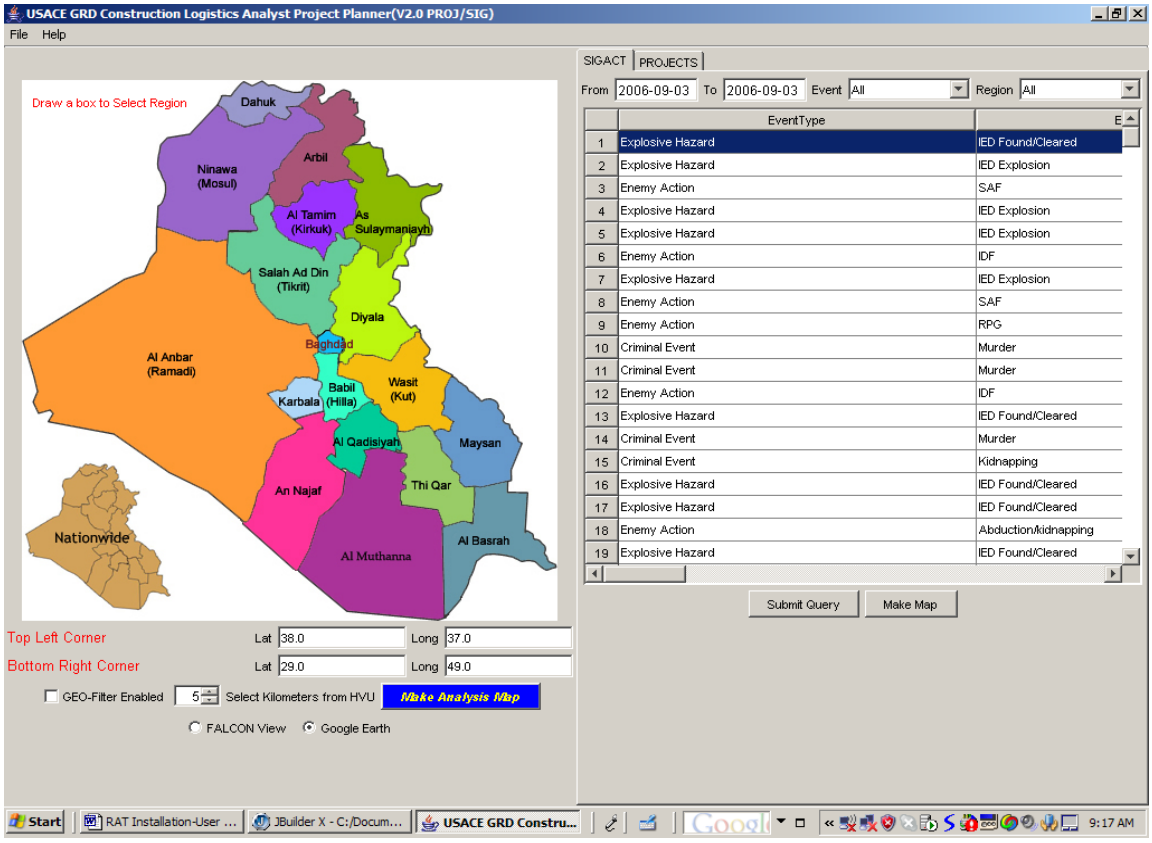

Figure 39. Reconstruction Analyst Toolset GUI

An example of the object output is shown below in Figure 5. The icon on each geo-spatial product has embedded meta-data.

Figure 40. Google Earth and Falcon View display

Reconstruction Analyst Toolset Objectives

- Rapidly display in a geo-spatial format the accurate position of Iraq Reconstruction related significant security events (SIGACTS), which impact reconstruction, and the location of projects.
- Allow the user to sort on various criteria to isolate specific types, geographic region and create a geo bounded box around a region to select objects inside.
- Allow user to select a project, which creates an overlay of only those significant attack (SIGACT's) within a user specified distance from the project.
- Allow users to display the data on various digital mapping products
- Operate on existing hardware and software operating systems
- Develop overlay capability to allow transfer of data to appropriate secure environment for expanded analysis of data at a higher classification level.
- Overlay Project data, SIGACTS and Reconstruction Team open source data to Analyze Security Impacts in an unclassified domain.
- Leverage existing capabilities and resources to automate data retrieval and presentation.

- Provides ability to compose view of disparate date sources on a common geo-spatial client, with ability to share analysis products with commercial partners

RAT is a Java Application developed in the J-Builder Integrated Development Environment. It interfaces a Microsoft Access Database via the JDBC-ODBC Bridge. The data resides on two tables labeled SIGACT and RMS. The SIGACT data originally is gathered from classified sources such as FUSION net, has certain data fields removed and is downgraded and placed on an Excel spreadsheet. The project data can be gathered from multiple sources to include the Army Corps of Engineers Reconstruction Management System, MAXIMO, or Iraq Reconstruction Management System. That spreadsheet can then be uploaded to the database and made available via the RAT. The System architecture diagram is in Figure 41, below:

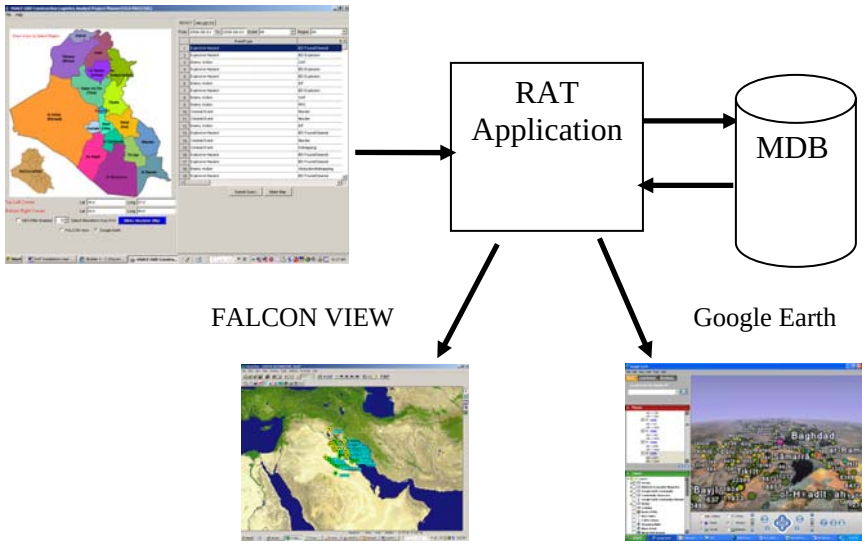

Figure 41. RAT System Overview

C. SOA PROTOTYPE SYSTEM

1. Overall Architecture

The web-service prototype employs SOA based technologies to implement a reference three tier data model. All services are developed in the Java programming language and use either the Enterprise Java Bean or Servlet construct. Two services reside in the *Report Tier* whose content is combined by a *fusion service* which then populates the results in the *Track Tier*. One report service provides content from a sample data set of Iraq reconstruction project data via an EJB to provide content for Simple Object Access Protocol (SOAP) transport and another via SERVLET to provide content via a Keyhole Markup Language (KML) for dynamic consumption by Google Earth. The second report level service provides content from a declassified sample representation of Iraq significant attack (SIGACT) database via an EJB to provide content for SOAP transport and another via SERVLET to provide content via a Keyhole Markup Language (KML) for dynamic consumption by Google Earth. Employing a BPEL service, the EJB report tier services are connected to a Fusion Service which conducts a simple calculation to determine how far each SIGACT is from a Reconstruction Project and takes the fused product of the two services and provides them to the new track tier service EJB. Note that both tiers are available for viewing and the services can be employed for use other than the BPEL orchestration. Figure 7 below shows the system architecture of the prototype.

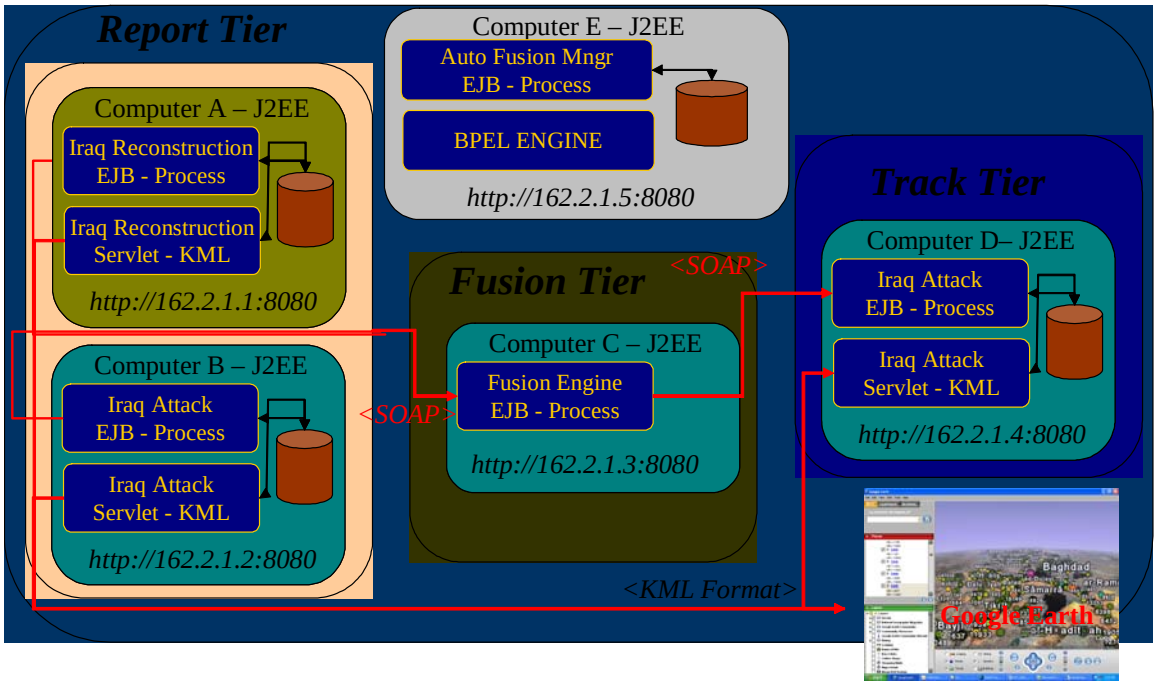

Figure 42. Web Service Prototype Architecture

Additionally, Figure 8 below shows the logic flow of the orchestration or BPEL data. The diagram is a product of Sun Microsystems NetBeans 5.5 developer software system. The bpel graphical designer shows the start of the process, the invocation of the first two or *report* level services, and the fusion engine consuming the data. Following the fusion engine processing, the BPEL service then takes the output and provides the content to the *Track Level* service to store. The BPEL process then reports completion and ends.

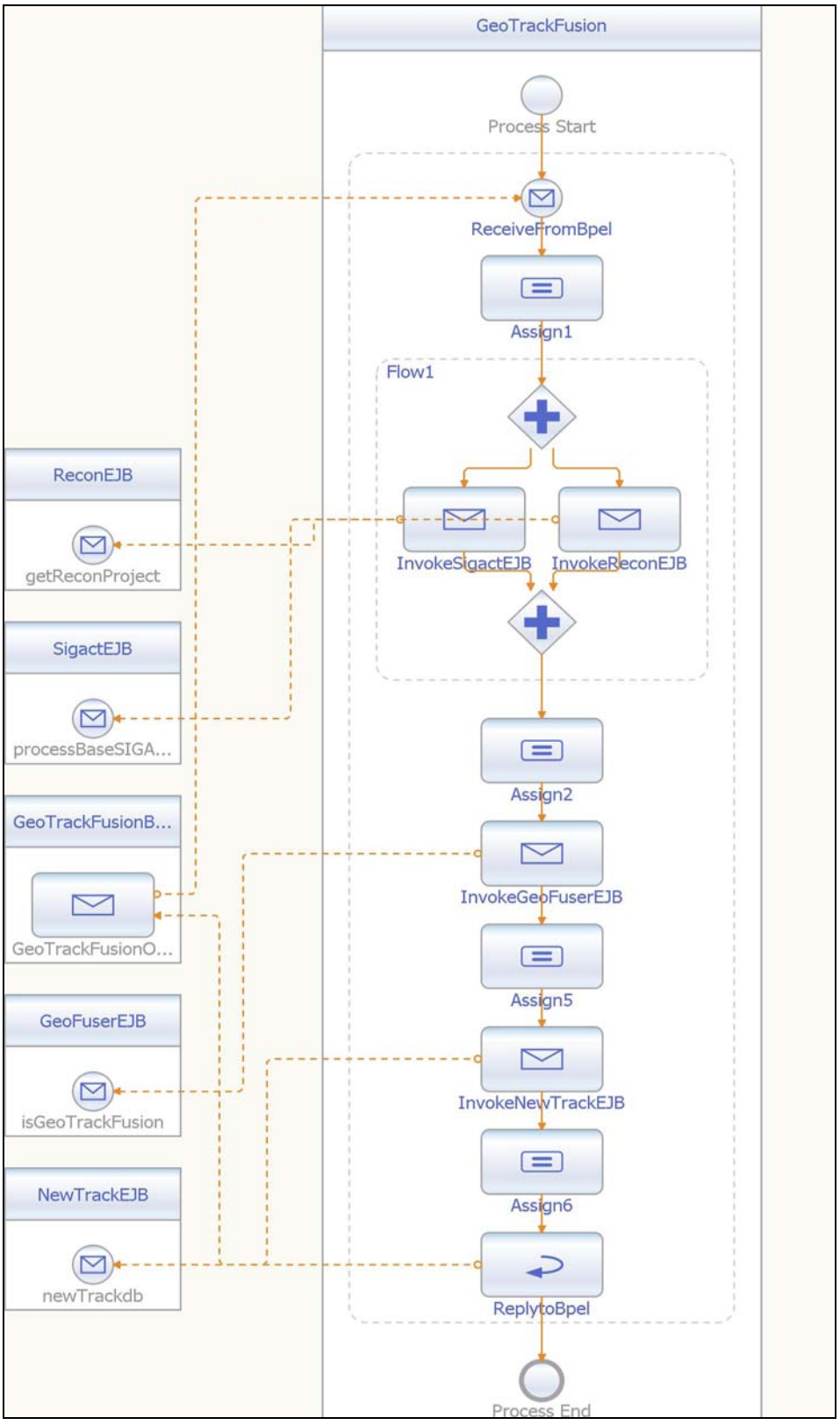

Figure 43. BPEL Process from NetBeans 5.5 GUI

2. System Environment and Foundation Commercial Products

The prototype system runs on Sun Application Server 9.0. The application server provides the foundation service hosting, orchestrations via the Business Process Application Language Engine (BPEL). The Google Earth servlet was developed in Java using the Boreland Java Development suite while the Enterprise Java Beans were developed using Sun's Netbeans 5.5 Integrated Development Environment. The code and services were developed as a demonstration prototype and lack appropriate error handling and testing associated with production systems. Although Sun Application Server was employed in development, the servlet's operated on BEA Weblogic Server 8.0 and Apache Web Server. The BPEL functionality was not employed on any other server than Sun Application 9.0. The persistence for the various services is provided by a Microsoft Access Database via the Object Database Connection (ODBC) to Java Database Connection (JDBC) Bridge.

3. Enterprise Java Bean (EJB) Modules

The prototype EJB services are simple web-enabled data modules that employ a Java Data Base Connection (JDBC) to retrieve and write data to a Microsoft Access Database. In this section we will discuss the web-service description language (WSDL) for each EJB, highlight the XML schema and detail critical processing logic. Actual prototype source code snippets will be shown in gray. Complete source code is provided in appendix A. Binary files can be obtained from the Naval Postgraduate School. A total of four Enterprise Java Beans are in the prototype; (1) SIGACT Processor, (2) Reconstruction Project Processor, (3) Fusion Engine Processor and (4) New Track Processor.

a. Significant Attack (SIGACT) Enterprise Java Bean (EJB) Module

Below is various components source code. First we will look at the XSD, then the WSDL and finally the EJB source code from the Sigact EJB. This servlet takes in two values named SIGACTStart and SIGACTFinish. These strings are date values which bound an SQL query into the data base. For example if a "7/15/06" start and

“8/15/06” date is entered then all the significant attacks in that month are returned. The XML schema (XSD) below has a red box which highlights two sections of the XSD. First are the two entering values discussed above with the two dates, the second are the output values named SIGACT which contain an attack location with a value for latitude (double) and longitude (double). Although not critical for the discussion of webservices at this point, developers might note that the database for sigacts contains objects with Military Grid Reference System (MGRS) vice Latitude and Longitude (LL) in degrees.seconds format required by many mapping systems. The conversion from MGRS to LL is part of the EJB java source code running on the server. If minimizing server load was a concern in a production system, batch conversion from MGRS to LL should be considered in a production system. The processSIGACT.xsd file is below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://BaseSigactEJB.sun.com/"
xmlns:tns="http://BaseSigactEJB.sun.com/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="processBaseSIGACTEJB"
type="tns:processBaseSIGACTEJB"/>
 <xs:element name="processBaseSIGACTEJBResponse"
type="tns:processBaseSIGACTEJBResponse"/>
<xs:complexType name="processBaseSIGACTEJB">
<xs:sequence>
<xs:element name="arg0" type="xs:string" minOccurs="0"/>
 <xs:element name="arg1" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="processBaseSIGACTEJBResponse">
<xs:sequence>
<xs:element name="return" type="tns:sigact" maxOccurs="unbounded"
minOccurs="0"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="sigact">
   <xs:sequence>
    <xs:element name="attackTarget" type="xs:string" minOccurs="0"/>
    <xs:element name="sigLat" type="xs:double"/>
    <xs:element name="sigLon" type="xs:double"/>
   </xs:sequence>
  </xs:complexType>
</xs:schema>
```

The next section of the EJB we will discuss is the Web-service Description Language (WSDL). Similar to the XSD file this is a formatted XML file as shown below to provide metadata to developers on how to employ the service. Three sections of processSIGACT.wsdl are highlighted and are discussed below.

```
<?xml version="1.0" encoding="UTF-8"?><definitions  <message
name="processBaseSIGACTEJB">
   <part name="parameters" element="tns:processBaseSIGACTEJB"/>
  </message>
  <message name="processBaseSIGACTEJBResponse">
   <part name="parameters" element="tns:processBaseSIGACTEJBResponse"/>
  </message>
  <portType name="BaseSigactEJB">
   <operation name="processBaseSIGACTEJB">
    <input message="tns:processBaseSIGACTEJB"/>
    <output message="tns:processBaseSIGACTEJBResponse"/>
   </operation>
  </portType>
  <binding name="BaseSigactEJBPortBinding" type="tns:BaseSigactEJB">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
   <operation name="processBaseSIGACTEJB">
    <soap:operation soapAction=""/>
    <input>
     <soap:body use="literal"/>
    </input>
    <output>
     <soap:body use="literal"/>
    </output>
   </operation>
  </binding>
  <service name="BaseSigactEJBService">
   <port name="BaseSigactEJBPort" binding="tns:BaseSigactEJBPortBinding">
    <soap:address location="http://localhost:8080/BaseSigactEJBService/BaseSigactEJB"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"/>
   </port>
  </service>
   <plnk:partnerLinkType name="BaseSigactEJBLinkType">
     <plnk:role name="BaseSigactEJBRole" portType="tns:BaseSigactEJB"/>
   </plnk:partnerLinkType>
</definitions>
```

First is the input and output messages derived from the schema file, second is the type of message. Document literal is employed in the above prototype software to maintain visibility of the data content and align the software as close to the open standards as possible. The third highlighted section contains the location of the service. Note the “localhost” location. The prototype can run on a number of locations as

long as the IP address and port is provided. To help an individual repeating the software "localhost" is employed to ease set up.

The final component of the Sigact EJB to discuss is the java source code. A number of functions are executed by the EJB however the critical ones include:

- Communicating with the SOAP messages
- Querying the database with the input values and obtaining the results of the query
- Converting the MGRS position data to a degrees and seconds Latitude and Longitude *double* format
- Returning the list of sigacts in a java array format to support a variable sized SOAP message to support return messages of variable sizes.

The first section highlighted employs the java @WebMethod class which processing the incoming SOAP message with the start and stop dates and returns the SOAP body with the SIGACT objects.

```
@Stateless()
@WebService()
public class BaseSigactEJB {

    Database database1 = new Database();
    QueryDataSet queryDataSet1 = new QueryDataSet();

    boolean stat = false;

   @WebMethod
   public List<sigact> processBaseSIGACTEJB(String sigactStart, String
sigactFinish) {

        return getSigBatch(sigactStart,sigactFinish);
   }
```

The next section of code queries the Microsoft access database and packages the results in an array format. In order to make the source readable some of the code is deleted. The complete source code is available in appendix A. The database query employs

Boreland© database query classes. Note the embedding of the start and finish strings into an SQL statement handled by the database query method and the invocation of the driver type.

```
public List<sigact> getSigBatch(String sigactStart, String sigactFinish){
        …
ueryDataSet1.setQuery(new
com.borland.dx.sql.dataset.QueryDescriptor(database1, "SELECT
SIGACT.EventType,SIGACT.EventCategory,SIGACT.OccuredWhen,SIGACT.Po
stedWhen,SIGACT.region,SIGAC.MGRS,SIGACT.PrimaryTarget,SIGACT.Targe
tCategory,SIGACT.City,
SIGACT.Province FROM
C:\\Documents and Settings\\kjrothen\\MyDocuments\\SIGACT\".SIGACT"
WHERE OccuredWhen BETWEEN #" + start + "# AND #" + finish + "# ", null,
true, Load.ALL));
database1.setConnection(new
com.borland.dx.sql.dataset.ConnectionDescriptor("jdbc:odbc:SIGACT", "", "",
false, "sun.jdbc.odbc.JdbcOdbcDriver"));
```

The next section of code creates the list object to return to the @webmethod class. Note the sections that create the array object, add the object to the list object and call the *Position* method which converts the MGRS data to Latitude and Longitude. As above, portions of the source are deleted to make the code readable and complete source code is available in Appendix A.

```
List<sigact> sigie = new ArrayList<sigact>();
    while (queryDataSet1.inBounds()) {…
       EventType = queryDataSet1.getString("EventType");
        MGRS = queryDataSet1.getString("MGRS");…
        Position R = MGRStoLat(MGR);
         userlat = R.getLat();
         userlon = R.getLng(); …
…
        sigact sig = new sigact();
         sig.setAttackTarget(PrimaryTarget);
         sig.setSigLat(userlat);
         sig.setSigLon(userlon);
         sigie.add(sig);
         queryDataSet1.next();
    }
```

The above code represents the sigact functionality of the prototype software. We will now examine the two other Enterprise Java Beans in the system; Reconstruction Project EJB and new track EJB.

b. Iraq Reconstruction Project Enterprise Java Bean (EJB) Module

Following the same format as the SIGACT EJB we will now examine the Reconstruction Project module source code and highlight critical processing logic. First we will look at the XSD, then the WSDL and finally the EJB source code from the ReconProject EJB. The reconstruction project takes in a single value named Sector, which maps to an investment sector as defined by the United States Army Corps of Engineers (USACE) leading the U.S. portion on the Iraq critical infrastructure reconstruction activities. The string object has values such as ELE to include Electricity, or H2O for water projects. The author has been unable to obtain a complete listing or data model for the values from USACE, however the sample data set contains only ELE, H2O, JUST, and SEC values. The sector value is used to conduct an SQL query, which returns all projects with that sector value.

The XML schema (XSD) below has a red box which highlights two critical sections of the XSD. First are the sector values which is a search field as

discussed above and the second are the output values named project which contain a projects name and location with a value for latitude (double) and longitude (double). Similar to SIGACT's the database for projects is also in Military Grid Reference System (MGRS) vice Latitude and Longitude (LL) in degrees.seconds format required by many mapping systems. The conversion from MGRS to LL is part of the EJB java source code running on the server. If minimizing server load was a concern in a production system, batch conversion from MGRS to LL should be considered in a production system. The ReconProjectEJBService_schema1 file is below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://ReconProjEJB.sun.com/"
xmlns:tns="http://ReconProjEJB.sun.com/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="getReconProject" type="tns:getReconProject"/>
 <xs:element name="getReconProjectResponse"
type="tns:getReconProjectResponse"/>
 <xs:complexType name="getReconProject">
 <xs:sequence>
 <xs:element name="Sector" type="xs:string" minOccurs="0"/>
 </xs:sequence>
  </xs:complexType>
  <xs:complexType name="getReconProjectResponse">
    <xs:sequence>
      <xs:element name="return" type="tns:retVal" maxOccurs="unbounded"
minOccurs="0"/>
  </xs:sequence>
  </xs:complexType>
  <xs:complexType name="retVal">
    <xs:sequence>
      <xs:element name="HVULat" type="xs:double"/>
      <xs:element name="HVULong" type="xs:double"/>
      <xs:element name="URI" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

The next section of the EJB we will discuss is the Web-service Description Language (WSDL). Similar to the XSD file this is a formatted XML file as

shown below to provide metadata to developers on how to employ the service. Three sections of ReconProjectEJB.wsdl are highlighted and are discussed below.

```
<?xml version="1.0" encoding="UTF-8"?><definitions
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://ReconProjEJB.sun.com/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
targetNamespace="http://ReconProjEJB.sun.com/" name="ReconProjectEJBService"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2004/03/partner-link/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://ReconProjEJB.sun.com/"
schemaLocation="ReconProjectEJBService_schema1.xsd"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"/>
    </xsd:schema>
  </types>
  <message name="getReconProject">
    <part name="parameters" element="tns:getReconProject"/>
  </message>
  <message name="getReconProjectResponse">
    <part name="parameters" element="tns:getReconProjectResponse"/>
  </message>
  <portType name="ReconProjectEJB">
    <operation name="getReconProject">
      <input message="tns:getReconProject"/>
      <output message="tns:getReconProjectResponse"/>
    </operation>
  </portType>
  <binding name="ReconProjectEJBPortBinding" type="tns:ReconProjectEJB">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getReconProject">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="ReconProjectEJBService">
    <port name="ReconProjectEJBPort" binding="tns:ReconProjectEJBPortBinding">
      <soap:address
location="http://localhost:8080/ReconProjectEJBService/ReconProjectEJB"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"/>
    </port>
  </service>
    <plnk:partnerLinkType name="ReconProjectEJBLinkType">
      <plnk:role name="ReconProjectEJBRole" portType="tns:ReconProjectEJB"/>
    </plnk:partnerLinkType>
</definitions>
```

First is the input and output messages derived from the schema file, second is the type of message. Document literal is employed in the above prototype software to maintain visibility of the data content and align the software as close to the open standards as possible. The third highlighted section contains the location of the service. Note the "localhost" location. The prototype can run on a number of locations as long as the IP address and port is provided. To help an individual repeating the software "localhost" is employed to ease set up.

The final component of the Reconstruction EJB to discuss is the java source code. A number of functions are executed by the EJB however the critical ones include:

- Communicating with the SOAP messages
- Querying the database with the input values and obtaining the results of the query
- Converting the MGRS position data to a degrees and seconds Latitude and Longitude *double* format
- Returning the list of reconstruction projects in a java array format to support a variable sized SOAP message to support return messages of variable sizes.

The first section highlighted employs the java @WebMethod class which processing the incoming SOAP message with the desired *sector* and returns the SOAP body with the Reconstruction Project objects.

```
@WebService
@Stateless
public class ReconProjectEJB{

     Database database1 = new Database();
    QueryDataSet queryDataSet1 = new QueryDataSet();
    DBDisposeMonitor dBDisposeMonitor1 = new DBDisposeMonitor();

@WebMethod
public List<RetVal> getReconProject(@WebParam(name="Sector") String
SECTOR){

      List<RetVal> projbatch = new ArrayList();
      return projbatch = getProjBatch(SECTOR);
  }
```

The next section of code queries the Microsoft access database and packages the results in an array format. In order to make the source readable some of the code is deleted. The complete source code is available in appendix A. The database query employs Boreland© database query classes. Note the embedding of the sector string into an SQL statement handled by the database query method and the invocation of the driver type.

```
public List<RetVal> getProjBatch(String SECTOR){
…
queryDataSet1.setQuery(new
com.borland.dx.sql.dataset.QueryDescriptor(database1, "SELECT
RMS_ALL.SHORT_NAME,RMS_ALL.URI,RMS_ALL.GRID_LOCATION" +
" FROM \"C:\\Documents and Settings\\kjrothen\\My
Documents\\RMS_ALL\".RMS_ALL"
 + " WHERE SECTOR = '"+SECTOR+ "' ", null, true, Load.ALL));
…
database1.setConnection(new
com.borland.dx.sql.dataset.ConnectionDescriptor("jdbc:odbc:RMS_ALL", "", "",
false, "sun.jdbc.odbc.JdbcOdbcDriver"));
```

The next section of code creates the list object to return to the @webmethod class. Note the sections that create the array object, add the object to the list object and call the *Position* method which converts the MGRS data to Latitude and Longitude. As above,

portions of the source are deleted to make the code readable and complete source code is available in appendix A.

```
List<RetVal> progie = new ArrayList<RetVal>();

      while (queryDataSet1.inBounds()) {

        MGRS = queryDataSet1.getString("GRID_LOCATION");
        String projectName = queryDataSet1.getString("URI");
        Position R = MGRStoLat(MGRS);
          double userlat = R.getLat();
          double userlon = R.getLng();
RetVal project = new RetVal();
        project.setURI(projectName);
        project.setHVULat(userlat);
        project.setHVULong(userlon);
        progie.add(project);
      queryDataSet1.next();
```

The above code represents the Reconstruction Project functionality of the prototype software. We will now examine the forth Enterprise Java Beans in the system; the track fusion EJB.

c. Fusion Engine Enterprise Java Bean (EJB) Module

We will now examine the Fusion Engine EJB in the prototype which takes the inputs from the SIGACT and Reconstruction Project EJB's and conducts a simple fusion algorithm. Both the inputs and output are provided via SOAP messages in document literal format. In the following section, the Fusion Enginer EJB source code and critical processing logic is reviewed. First we will look at the XSD, then the WSDL and finally the EJB source code from the GeoTrackFusionEJB. Change it to 4th order paragraph – check all the way through

The XML schema (XSD) below has a red box which highlights two critical sections of the XSD. The first box highlights the input messages; one for the SIGACTs which includes the latitude, longitude and attack type and another message for reconstruction projects which include latitude, longitude and project name for each PROJECT object. The second box highlights the output message which has the

combined or “fused” object to include the latitude, longitude and combined name (in the case of the prototype the combined name is a string concatenation of the attack type and the project name). The GeoTrackFusionService_schema1 file is below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0"
targetNamespace="http://GeoTrackFusion.sun.com/"
xmlns:tns="http://GeoTrackFusion.sun.com/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="isGeoTrackFusion" type="tns:isGeoTrackFusion"/>
<xs:element name="isGeoTrackFusionResponse"
type="tns:isGeoTrackFusionResponse"/>
<xs:complexType name="isGeoTrackFusion">
<xs:sequence>
<xs:element name="arg0" type="tns:sigact" maxOccurs="unbounded"
minOccurs="0"/>
<xs:element name="arg1" type="tns:project" maxOccurs="unbounded"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="sigact">
<xs:sequence>
<xs:element name="attackTarget" type="xs:string" minOccurs="0"/>
<xs:element name="sigLat" type="xs:double"/>
<xs:element name="sigLon" type="xs:double"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="project">
<xs:sequence>
<xs:element name="projLat" type="xs:double"/>
    <xs:element name="projLon" type="xs:double"/>
    <xs:element name="projectName" type="xs:string" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="isGeoTrackFusionResponse">
  <xs:sequence>
    <xs:element name="return" type="tns:combined"
maxOccurs="unbounded" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="combined">
  <xs:sequence>
    <xs:element name="combLat" type="xs:double"/>
    <xs:element name="combLon" type="xs:double"/>
    <xs:element name="combName" type="xs:string" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

The next section of the EJB we will discuss is the Web-service Description Language (WSDL). Similar to the XSD file this is a formatted XML file as shown below to provide metadata to developers on how to employ the service. Three sections of ReconProjectEJB.wsdl are highlighted. First is the input and output messages derived from the schema file. The second box highlights the type of message, in this case Document literal is employed in the above prototype software to maintain visibility of the data content and align the software as close to the open standards as possible. The third highlighted section contains the location of the service. Note the “localhost” location. The prototype can run on a number of locations as long as the IP address and port is provided. To help an individual repeating the software “localhost” is employed to ease set up.

```
<?xml version="1.0" encoding="UTF-8"?><definitions
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://GeoTrackFusion.sun.com/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
targetNamespace="http://GeoTrackFusion.sun.com/" name="GeoTrackFusionService"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2004/03/partner-link/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://GeoTrackFusion.sun.com/"
schemaLocation="GeoTrackFusionService_schema1.xsd"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"/>
    </xsd:schema>
  </types>
  <message name="isGeoTrackFusion">
    <part name="parameters" element="tns:isGeoTrackFusion"/>
  </message>
  <message name="isGeoTrackFusionResponse">
    <part name="parameters" element="tns:isGeoTrackFusionResponse"/>
  </message>
  <portType name="GeoTrackFusion">
    <operation name="isGeoTrackFusion">
      <input message="tns:isGeoTrackFusion"/>
      <output message="tns:isGeoTrackFusionResponse"/>
    </operation>
  </portType>
  <binding name="GeoTrackFusionPortBinding" type="tns:GeoTrackFusion">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="isGeoTrackFusion">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="GeoTrackFusionService">
    <port name="GeoTrackFusionPort" binding="tns:GeoTrackFusionPortBinding">
      <soap:address
location="http://localhost:8080/GeoTrackFusionService/GeoTrackFusion"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"/>
    </port>
  </service>
    <plnk:partnerLinkType name="GeoTrackFusionLinkType">
      <plnk:role name="GeoTrackFusionRole" portType="tns:GeoTrackFusion"/>
```

The final component of the Fusion Engine EJB to discuss is the java source code. A number of functions are executed by the EJB however the critical ones include:

- Communicating with the SOAP messages
- Executing the fusion engine algorithm
- Returning the list of new combined objects in a java array format to support a variable sized SOAP message to support return messages of variable sizes.

The first section highlighted employs the java @WebMethod class which processing the incoming SOAP messages from the SIGACT and Reconstruction Project EJB's.

```
@Stateless()
@WebService()
public class GeoTrackFusion {

@WebMethod
public List<combined> isGeoTrackFusion(List<sigact> sigbatch, List<project>
projbatch){

      List<combined> fusebatch = new ArrayList();
      fusebatch = getfusedBatch(sigbatch,projbatch);
return fusebatch;


}
```

The next section of code executes the fusion algorithm and packages the results in an array format. In order to make the source readable some of the code is deleted. The complete source code is available in appendix A. The first highlighted box shows the call to the next method which evaluates if the SIGACT and Reconstruction Project objects are next to each other. If the *object* returns a true value an average of the two objects is created to approximate the position of the new or "combined" object as shown in the second highlighted box.

```
public List<combined> getfusedBatch(List<sigact> sigbatch, List<project>
projbatch) {

      List<combined> comie = new ArrayList<combined>();
   …
       Iterator<sigact> sigIter = sigbatch.iterator();
      while(sigIter.hasNext()) {
         sigact s = sigIter.next();
         siglat = s.getSigLat();
         siglon = s.getSigLon();
         Iterator<project> projIter = projbatch.iterator();
         while(projIter.hasNext()) {
            project p = projIter.next();
            projlat = p.getProjLat();
            projlon = p.getProjLon();
            // maketrack fourth varialble is a distance setting in Miles.
            maketrack = isnextTo(projlat,projlon,siglat,siglon,5);
            if (maketrack){
               newname = p.getProjectName() + s.getAttackTarget();

               double newlat = (projlat + siglat)/2;
               double newlon = (projlon + siglon)/2;

                 combined com = new combined();
                 com.setcombName(newname);
                 com.setcombLat(newlat);
                 com.setcombLon(newlon);
                 comie.add(com);
              }
              return comie;
```

The above code represents the Fusion Engine functionality of the prototype software. We will now examine the last Enterprise Java Beans in the system; the new track EJB.

d. New Track Enterprise Java Bean (EJB) Module

The last EJB in the prototype writes the fused objects from the fusion engine EJB into a database. We will examine the New Track module source code and highlight critical processing logic. First we will look at the XSD, then the WSDL and finally the EJB source code from the NewTrackService EJB. The New Track Service takes in the SOAP message with a series of objects with a name, latitude and longitude.

The service then returns a string reporting it completed writing the objects to its database. The XML schema (XSD) below has a red box which highlights two critical sections of the XSD. First is the input message with the combined objects. The second is the return object which is a simple string reporting completion of the database write. The NewTrackServiceService_schema1 file is below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0"
targetNamespace="http://NewTrackService.sun.com/"
xmlns:tns="http://NewTrackService.sun.com/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="newTrackdb" type="tns:newTrackdb"/>
 <xs:element name="newTrackdbResponse"
type="tns:newTrackdbResponse"/>
 <xs:complexType name="newTrackdb">
 <xs:sequence>
 <xs:element name="arg0" type="tns:combined" maxOccurs="unbounded"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="combined">
<xs:sequence>
   <xs:element name="combLat" type="xs:double"/>
   <xs:element name="combLon" type="xs:double"/>
   <xs:element name="combName" type="xs:string" minOccurs="0"/>
 </xs:sequence>
 </xs:complexType>
 <xs:complexType name="newTrackdbResponse">
<xs:sequence>
   <xs:element name="return" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

The next section of the EJB we will discuss is the Web-service Discription Language (WSDL). Similar to the XSD file this is a formatted XML file as shown below to provide metadata to developers on how to employ the service. Three sections of NewTrackService.wsdl are highlighted and are discussed below.

```
<?xml version="1.0" encoding="UTF-8"?><definitions
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://NewTrackService.sun.com/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
targetNamespace="http://NewTrackService.sun.com/" name="NewTrackServiceService"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2004/03/partner-link/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://NewTrackService.sun.com/"
schemaLocation="NewTrackServiceService_schema1.xsd"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"/>
    </xsd:schema>
  </types>
  <message name="newTrackdb">
    <part name="parameters" element="tns:newTrackdb"/>
  </message>
  <message name="newTrackdbResponse">
    <part name="parameters" element="tns:newTrackdbResponse"/>
  </message>
  <portType name="NewTrackService">
    <operation name="newTrackdb">
      <input message="tns:newTrackdb"/>
      <output message="tns:newTrackdbResponse"/>
    </operation>
  </portType>
  <binding name="NewTrackServicePortBinding" type="tns:NewTrackService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="newTrackdb">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="NewTrackServiceService">
    <port name="NewTrackServicePort" binding="tns:NewTrackServicePortBinding">
      <soap:address
location="http://localhost:8080/NewTrackServiceService/NewTrackService"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"/>
    </port>
  </service>
    <plnk:partnerLinkType name="NewTrackServiceLinkType">
      <plnk:role name="NewTrackServiceRole" portType="tns:NewTrackService"/>
    </plnk:partnerLinkType>
```

First is the input and output messages derived from the schema file, second is the type of message. Document literal is employed in the above prototype software to maintain visibility of the data content and align the software as close to the open standards as possible. The third highlighted section contains the location of the service. Note the "localhost" location. The prototype can run on a number of locations as long as the IP address and port is provided. To help an individual repeating the software "localhost" is employed to ease set up.

The final component of the NewTrackService EJB to discuss is the java source code. A number of functions are executed by the EJB however the critical ones include:

- Communicating with the SOAP messages
- Writing the database with the new objects
- Returning a statement that the operation is complete via a SOAP message

The first section highlighted employs the java @WebMethod class which processing the incoming SOAP message with the new track objects.

```
@WebService
@Stateless
public class NewTrackService{
   @WebMethod
public String newTrackdb(List<combined> fusebatch){



   conn = DriverManager.getConnection("jdbc:odbc:COMBINED");
;
```

The next section of code highlights the writing to the Microsoft access database. In order to make the source readable some of the code is deleted. The complete source code is available in appendix A. In this case the database write does not employ Boreland© database query classes, but uses Java extensions to demonstrate that other methods are

available. Note the embedding of the objects into an INSERT SQL statement handled by the database query method and the invocation of the driver type.

```
try{
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
} catch(java.lang.ClassNotFoundException e) {
   System.err.print("ClassNotFoundException: ");
   System.err.println(e.getMessage());
}
…. Iterator<combined> comIter = fusebatch.iterator();
        while(comIter.hasNext()) {
           combined c = comIter.next();
           comlat = c.getcombLat();
           comlon = c.getcombLon();
           comname = c.getcombName();
   Statement stmt;
          try {
             stmt = conn.createStatement();
         stmt.executeUpdate("INSERT INTO Combined VALUES
("'+comname+'","+comlat+","+comlon+")");
          } catch (SQLException ex) {
             ex.printStackTrace();
          }
        }
```

The above code represents the Reconstruction Project functionality of the prototype software. Next we will examine the Java Servlet technology employed by the prototype to expose the services to the Google Earth mapping software in the KML format.

4. Servlet Modules with Keyhole Markup Language

Significant debate surrounds the use of Google Earth as a tactical mapping/visualization system. However, Google Earth has a number of attributes making it attractive for use in the prototype. Firstly, it's free and generally ubiquitous. Many users are familiar with the product and how to employ it as a user. Secondly, the interface methods are simple and utilize web service technology to communicate, making it an excellent candidate to visualize objects. The downside to Google Earth includes the bandwidth required to communicate with the GE server and the vulnerability associated with relying exclusively on a commercial entity to provide mapping software. Despite

the obstacles associated with Google Earth, GE combined with the legacy software developed by the author and described earlier in this section provided a suitable capability working with Iraqi Engineers and private security entities that did not have a clearance to use DoD mapping software. Additionally, Google Earth provides a good environment to validate the *three tier model* associated with this dissertation.

Google provides extensive documentation on how to communicate with Google Earth via its Keyhole Markup Language (KML). KML is a XML based schema that allows the developed to describe an object that Google Earth can display as an overlay in its mapping software. Two general methods are employed to communicate with Google Earth. One is a static file download, where a KML file is opened by a GE user and the overlay is loaded into GE. The second method is a dynamic connection between a servlet and a Google Earth client instance. The first method is employed by the traditional application developed by the author; the second dynamic method is employed by the prototype software. Figure 9 below shows the general architecture of the dynamic system.

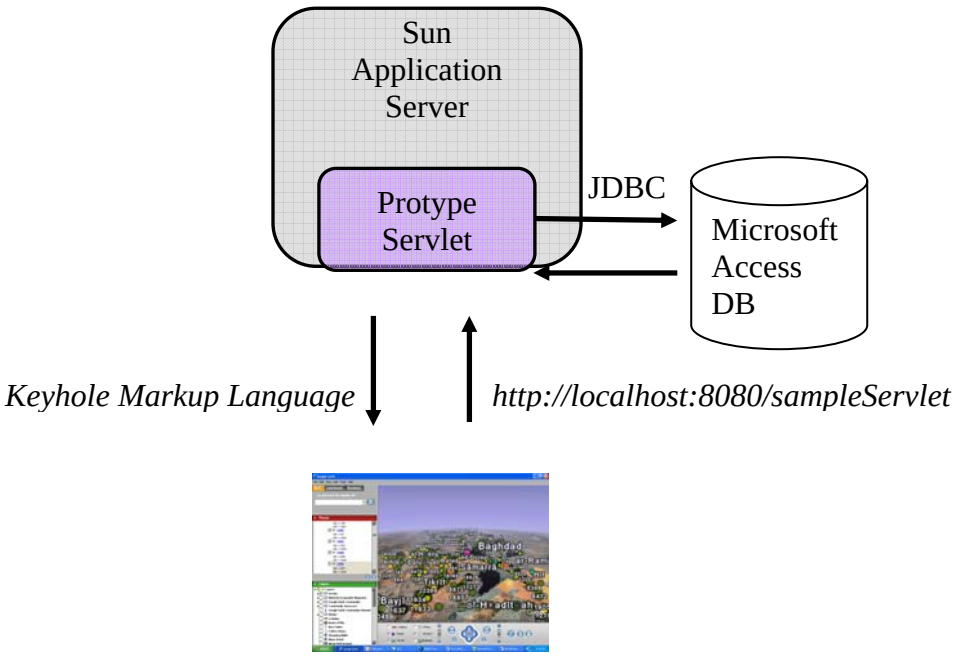

Figure 44. Dynamic Google Earth System Architecture

Three servlets are employed in the prototype to provide KML formatted data to Google Earth. These servlets are;

- A Significant Attack (SIGACT) servlet that shows the attacks over a given period.
- A Reconstruction Project servlet that shows the reconstruction projects for a given infrastructure category (i.e. Justice, Water, Defense, Education)
- A New Track servlet that shows the new tracks created by the fused product of the SIGACT and Reconstruction Project servlets.

The next section will detail those three servlets.

5. SIGACT KML Servlet

The first servlet we will review provides a dynamic KML formatted message to Google Earth. The architecture employs a ReST model and does not use SOAP messages but rather sends the data directly over HTTP using the doGet and doPost java methods. We will examine two files for the servlet. One is the KML file that Google Earth opens that provides the location and refresh rate of the servlet, the other is the servlet source code. Similar to the previous sections only critical portions of the source code are placed in the gray boxes with all the source code available in appendix A and binaries available from the Naval Postgraduate School.

The XML file below is opened by Google Earth to direct the client to interface with a servlet to obtain overlays. The first highlighted section provides the location of the servlet. As in previous sections, localhost is employed to assist in duplicating the prototype, however any IP address or port can be employed. Note the two parameters embedded in the URL. They include a start and stop date. Those dates are used by the servlet to obtain the period desired by the user. The second highlighted section in a red box shows the desired refresh rate. In this case the 45 seconds, directs the client to execute a refresh from the servlet at the determine interval.

```
<?xml version="1.0" encoding="UTF-8"?>
<NetworkLink>
  <description>Track Updates Every 45 Seconds</description>
  <name>Significent Attack Postion Feed</name>
  <visibility>0</visibility>
  <url>http://localhost:8080/sigactKMLV3/learnerServ?start=07-01-
2000&amp;amp;finish=07-01-2008</url>
  <refreshPeriod>45</refreshPeriod>
  <refreshVisibility>1</refreshVisibility>
</NetworkLink>
```

Next let's examine the Java Servlet source code. The servlet conducts a number of functions to include:

- Performing the doGet function and receive the parameters from the URL
- Conduct an SQL query of the SIGACT database for the period prescribed by the parameters
- Convert the results of the SQL query into a KML formatted message
- Push the results to the Google Earth client employing the doGet method

We will examine the major processing logic in the below section.

The first section of source code highlighted below shows the *doGet(HttpServlet request and response)* method. Note the start and finish parameters are called using a request.getParameter call that looks for the *start* and *finish* string variables. Also note the CONTENT_TYPE output format of text/html. This sets the output of the servlet response.

```
public class sigactMLV3 extends HttpServlet {
  private static final String CONTENT_TYPE = "text/html";
  Database database1 = new Database();
  QueryDataSet queryDataSet1 = new QueryDataSet();
  DBDisposeMonitor dBDisposeMonitor1 = new DBDisposeMonitor();

  //Process the HTTP Get request
  public void doGet(HttpServletRequest request, HttpServletResponse response)
throws      ServletException, IOException {

     String start = request.getParameter("start");
     String finish = request.getParameter("finish");

     Timestamp OccuredWhen;
   String EventType = "begin";
     String Name = "null";
     String Relative = "null";
     String Location = "null";
     String MGRS = "null";
….
```

The next section of source code will examine the database query action. In the case of the KML servlets we employ the boreland© database methods to conduct the queries. The first highlighted section shows the embedding of the start and finish dates in the SQL query. The second highlighted section shows the database interface driver.

```
queryDataSet1.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(database1,
"SELECT
SIGACT.EventType,SIGACT.EventCategory,SIGACT.OccuredWhen,SIGACT.Poste
dWhen,SIGACT.region,SIGACT.MGRS,SIGACT.PrimaryTarget,SIGACT.TargetCat
egory,SIGACT.City,SIGACT.Province FROM \"C:\\Documents and
Settings\\kjrothen\\My Documents\\SIGACT\".SIGACT"
  + " WHERE OccuredWhen BETWEEN #" + start + "# AND #" + finish + "# ", null,
true, Load.ALL));

database1.setConnection(new
com.borland.dx.sql.dataset.ConnectionDescriptor("jdbc:odbc:SIGACT", "", "", false,
"sun.jdbc.odbc.JdbcOdbcDriver"));
```

The final section we will examine is the KML formatting code. The KML format is documented via Google Earth's tutorial [GOOGLE EARTH], however some key elements are highlighted below. The top section of the KML file is a header which describes attributes such as icon and label style, the second portion of the KML file is a repeating element where each object is described. For the file shown below, first note the icon and label style descriptions, which provide the Google Earth client with icon and label font information for each object. The second highlighted section shows the procedure employed to write the output of the database to the KML document. The prototype uses an while loop based on remaining objects in the query set. The third highlighted section shows where a conversion from Military Grid Reference System to latitude and longitude is executed using the *position* method. The EJB section has a more detailed explanation of the MGRS to LL conversion, however the source code is adapted from National Geo-spatial Agency (NGA) source code available via the NGA site and employed by the eXtensible Tactical C4I Framework (XTCF) project at SPAWAR San Diego labs. The final section highlighted is where the latitude and longitude position is placed in the KML file.

```
  out.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
      out.write("<kml xmlns=\"http://earth.google.com/kml/2.0\">\n");
      out.write("<Document>\n");
      out.write("<name> USACE Map Creater (SIGACT)</name>\n");
      out.write("<open>1</open>\n");
      out.write("<description><![SAMPLE DATA]></description>\n");
      out.write("<Style id=\"Standard\">");
      out.write("   <IconStyle>\n");
      out.write("    <color>ff0000ff</color>\n");
      out.write("    <Icon>\n");
      out.write("    <href>root://icons/palette-4.png</href>\n");
      ….
      out.write("</IconStyle>\n");
      out.write("<LabelStyle>");
…..
      out.write("<name>SIGACT " + start + " to " + finish + "</name>\n");
      out.write("<open>0</open>\n");
      out.write("/Folder>\n");

      queryDataSet1.first();

      while (queryDataSet1.inBounds()) {
       ++mapcountstart;
      // String style = "";
       EventType = queryDataSet1.getString("EventType");
       String EventCategory = queryDataSet1.getString("EventCategory");
       OccuredWhen = queryDataSet1.getTimestamp("OccuredWhen");

       MGRS = queryDataSet1.getString("MGRS");
       String MGR = MGRS.replaceAll(" ", "");
       Position R = MGRStoLat(MGR);

        userlat = R.getLat();
        userlon = R.getLng();

      String klmString = "<Placemark>\n"
        + "<name>"+ EventCategory+ "</name>\n"
        +"<description><![CDATA[Name-"+ EventType + "<br /> Occured-" +
        OccuredWhen + "<br /> Target-" + PrimaryTarget +…
        + "<styleUrl>#Standard</styleUrl>\n"
           + "<coordinates>"+userlon+","+userlat+"</coordinates>\n"
       + "</Point>\n"
           + "</Placemark>\n";
        out.write(klmString);
```

This section documented the SIGACT servlet, we will next examine the Reconstruction Project Servlet.

6. Reconstruction Project KML servlet.

The second servlet we will review provides a dynamic KML formatted message to Google Earth. The architecture employs a ReST model and does not use SOAP messages but rather sends the data directly over HTTP using the doGet and doPost java methods. We will examine two files for the servlet. One is the KML file that Google Earth opens that provides the location and refresh rate of the servlet, the other is the servlet source code.

The XML file below is opened by Google Earth to direct the client to interface with a servlet to obtain overlays. The first highlighted section provides the location of the servlet. As in previous sections, localhost is employed to assist in duplicating the prototype, however any IP address or port can be employed. Note the single parameter, sector embedded in the URL. The sector variable is used by the servlet to obtain the reconstruction sector desired by the user, in this case the electric sector "ELEC" is desired. The second highlighted section in a red box shows the desired refresh rate. In this case the 45 seconds, directs the client to execute a refresh from the servlet at the determine interval.

```
<?xml version="1.0" encoding="UTF-8"?>
<NetworkLink>
  <description>Track Updates Every 45 Seconds</description>
  <name>Iraq Reconstruction Projects</name>
  <visibility>0</visibility>
  <url>http://localhost:8080/iraqdemoproj/iraqdemoprojsev?sector=ELEC</url>
  <refreshPeriod>45</refreshPeriod>
  <refreshVisibility>1</refreshVisibility>
</NetworkLink>
```

Next let's examine the Java Servlet source code. The servlet conducts a number of functions to include:

- Performing the doGet function and receive the parameters from the URL

- Conduct an SQL query of the RECON database for the sector prescribed by the parameter.
- Convert the results of the SQL query into a KML formatted message
- Push the results to the Google Earth client employing the doGet respond method

The first section of source code highlighted below shows the *doGet(HttpServlet request and response)* method. Note the sector parameter is called using a request.getParameter call that looks for the sector string variable. Also note the CONTENT_TYPE output format of text/html. This sets the output of the servlet response.

```
public class reconKML extends HttpServlet {
  private static final String CONTENT_TYPE = "text/html";
  Database database1 = new Database();
  QueryDataSet queryDataSet1 = new QueryDataSet();
  DBDisposeMonitor dBDisposeMonitor1 = new DBDisposeMonitor();
…
  //Process the HTTP Get request
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    String sector = request.getParameter("sector");
    Timestamp OccuredWhen;
       String ProjName = "begin";
       String tempQuery = "null";
       String ProjectID = "null";
       String Contractor = "null";
       String ExecutingAgency = "null";
       ….
```

The next section of source code will examine the database query action. In the case of the KML servlets we employ the boreland© database methods to conduct the queries. The first highlighted section shows the embedding of the sector variable in the SQL query. The second highlighted section shows the database interface driver.

```
  queryDataSet1.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(database1,
"SELECT RMS_ALL.SHORT_NAME,RMS_ALL.URI,RMS_ALL.GRID_LOCATION" +
  " FROM \"C:\\Documents and Settings\\kjrothen\\My Documents\\RMS_ALL\".RMS_ALL" +
       " WHERE SECTOR = '"+sector+ "' ", null, true, Load.ALL));

  database1.setConnection(new
com.borland.dx.sql.dataset.ConnectionDescriptor("jdbc:odbc:RMS_ALL", "", "", false,
"sun.jdbc.odbc.JdbcOdbcDriver"));
  database1.setUseCaseSensitiveId(true);
  database1.setUseSpacePadding(false);
  database1.setDatabaseName("");
   queryDataSet1.open(); …
```

The final section we will examine is the KML formatting code. The KML format is documented via Google Earth's tutorial [GOOGLE EARTH], however some key elements are highlighted below. The top section of the KML file is a header which describes attributes such as icon and label style, the second portion of the KML file is a repeating element where each object is described. For the file shown below first note the icon and label style descriptions, which direct the Google Earth client which icon and label font to display for each object. The second highlighted section shows the procedure employed to write the output of the database to the KML document. The prototype uses a while loop based on remaining objects in the query set. The third highlighted section shows where a conversion from Military Grid Reference System to latitude and longitude is executed using the *position* method. The EJB section has a more detailed explanation of the MGRS to LL conversion, however the source code is adapted from National Geo-spatial Agency (NGA) source code available via the NGA site and employed by the eXtensible Tactical C4I Framework (XTCF) project at SPAWAR San Diego labs. The final section highlighted is where the latitude and longitude position is placed in the KML file.

```
out.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
        out.write("<kml xmlns=\"http://earth.google.com/kml/2.0\">\n");
        out.write("<Document>\n");
        out.write("<name> USACE Map Creater</name>\n");
        out.write("<open>1</open>\n");
        out.write(
           "<description><![CDATA[SAMPLE/]></description>\n");
        out.write("<Style id=\"ELE\">");
        out.write("   <IconStyle>\n");
        out.write("    <color>FFFF55FF</color>\n");
        out.write("    <Icon>\n");
        out.write("    <href>root://icons/palette-4.png</href>\n");
       out.write("<LabelStyle>");
        out.write("    <color>ffb3cefc</color>\n");
        out.write("<scale>1.1</scale>");
        out.write("</LabelStyle>");
        queryDataSet1.first();

        while (queryDataSet1.inBounds()) {

          String style = "";
          ProjName = queryDataSet1.getString("SHORT_NAME");
          String MGRS = queryDataSet1.getString("GRID_LOCATION");

          Position R = MGRStoLat(MGRS);

          userlat = R.getLat();
          userlon = R.getLng();

      String klmString = "<Placemark>\n"

            + "<styleUrl>" + style + "</styleUrl>\n"
            + "<Point>\n"
            + "<extrude>0</extrude>\n"
            + "<altitudeMode>clampedToGround</altitudeMode>\n"
            + "<coordinates>" + userlon + "," + userlat +
            "</coordinates>\n"
            + "</Point>\n"
            + "</Placemark>\n";
```

This section documented the Reconstruction Project servlet, we will next examine the New Track servlet.

7. New Track KML servlet.

The last servlet we will review provides a dynamic KML formatted message to Google Earth with the new tracks created from the fused content of the SIGACT and Reconstruction Project. The architecture employs a ReST model and does not use SOAP messages but rather sends the data directly over HTTP using the doGet and doPost java methods. We will examine two files for the servlet. One is the KML file that Google Earth opens that provides the location and refresh rate of the servlet, the other is the servlet source code.

The XML file below is opened by Google Earth to direct the client to interface with a servlet to obtain overlays. The first highlighted section provides the location of the servlet. As in previous sections, localhost is employed to assist in duplicating the prototype, however any IP address or port can be employed. Note in this case no parameters are embedded in the URL. One can imagine a case where a parameter such as type of track, friend lies or hostiles, or any number of criteria could be employed, however for this servlet none were required. The second highlighted section in a red box shows the desired refresh rate. In this case the 45 seconds, directs the client to execute a refresh from the servlet at the determine interval.

```
<?xml version="1.0" encoding="UTF-8"?>
<NetworkLink>
  <description>Track Updates Every 45 Seconds</description>
  <name>Fused Track Feed</name>
  <visibility>0</visibility>
  <url>http://localhost:8080/combinedKML/learnerServ</url>
  <refreshPeriod>45</refreshPeriod>
  <refreshVisibility>1</refreshVisibility>
</NetworkLink>
```

Next let's examine the Java Servlet source code. The servlet conducts a number of functions to include:

- Performing the doGet function and receive the parameters from the URL
- Conduct an SQL query of the Combined database for all the objects.
- Convert the results of the SQL query into a KML formatted message
- Push the results to the Google Earth client employing the doGet method

We will examine the major processing logic in the below section.

The first section of source code highlighted below shows the *doGet(HttpServlet request and response)* method. Note that no parameters are called using request.getParameter. Also note the CONTENT_TYPE output format of text/html. This sets the output of the servlet response.

```
public class combinedKML extends HttpServlet {
  private static final String CONTENT_TYPE = "text/html";
  Database database1 = new Database();
  QueryDataSet queryDataSet1 = new QueryDataSet();
  DBDisposeMonitor dBDisposeMonitor1 = new DBDisposeMonitor();

  //Process the HTTP Get request
  public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

      int mapcountstart = 0;
      double userlon = 0;
      double userlat = 0;
           int i = 1;
..
```

The next section of source code will examine the database query action. In the case of the KML servlets we employ the boreland© database methods to conduct the queries. The first highlighted section shows the SQL query. The second highlighted section shows the database interface driver.

```
queryDataSet1.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(database1,
"SELECT Combined.Name,Combined.ComLat,Combined.ComLon" +
            " FROM \"C:\\Documents and Settings\\kjrothen\\My
Documents\\combined\".Combined", null, true, Load.ALL));
            database1.setConnection(new
com.borland.dx.sql.dataset.ConnectionDescriptor("jdbc:odbc:combined", "", "", false,
"sun.jdbc.odbc.JdbcOdbcDriver"));
```

The final section we will examine is the KML formatting code. The KML format is documented via Google Earth's tutorial [GOOGLE EARTH], however some key elements are highlighted below. The top section of the KML file is a header which describes attributes such as icon and label style, the second portion of the KML file is a repeating element where each object is described. For the file shown below first note the icon and label style descriptions, which direct the Google Earth client which icon and label font to display for each object. The second highlighted section shows the procedure employed to write the output of the database to the KML document. The prototype uses a while loop based on remaining objects in the query set. The final section highlighted is where the latitude and longitude position is placed in the KML file.

```
  out.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
      out.write("<kml xmlns=\"http://earth.google.com/kml/2.0\">\n");
      out.write("<Document>\n");
      out.write("<name> Combined Track Objects</name>\n");
      out.write("<open>1</open>\n");
      out.write("<description><![CDATA[ UNCLASS//FOUO//
]]></description>\n");
      out.write("<Style id=\"Standard\">");
      out.write("   <IconStyle>\n");
      out.write("    <color>ff00aa00</color>\n");
      out.write("    <Icon>\n");
      out.write("    <href>root://icons/palette-4.png</href>\n");
      out.write("   <x>32</x>\n");
      out.write("   <y>128</y>\n");
      out.write("   <w>32</w>\n");
      out.write("   <h>32</h>\n");
      out.write("  </Icon>\n");
      out.write("</IconStyle>\n");
      out.write("<LabelStyle>");
      out.write("    <color>ff0000ff</color>\n");
      out.write("<scale>1.1</scale>");
      out.write("</LabelStyle>");
      out.write("</Style>\n");
      out.write("<colorMode>normal</colorMode>");
      QueryDataSet1.first();

      while (queryDataSet1.inBounds()) {
       ++mapcountstart;
      // String style = "";
       String Name = queryDataSet1.getString("Name");
       userlat = queryDataSet1.getDouble("ComLat");
       userlon = queryDataSet1.getDouble("ComLon");

       String klmString = "<Placemark>\n"
        + "<name>"+ Name + "</name>\n" +
       // +"<description><![CDATA[Name-"+ EventType + "<br /> Occured-" +
       // OccuredWhen + "<br /> Target-" + PrimaryTarget +

       "<styleUrl>#Standard</styleUrl>\n"
        + "<Point>\n"
           + "<extrude>0</extrude>\n"
           + "<altitudeMode>clampedToGround</altitudeMode>\n"
           + "<coordinates>"+userlon+","+userlat+"</coordinates>\n"
       + "</Point>\n"
           + "</Placemark>\n";
```

This section documented the New Track Servlet, next we will examine the Business Process Execution Language segments of the prototype software.

8. Business Process Execution Language Modules (BPEL)

Business Process Execution Language for web-services (BPEL4WS) is an open WS standard for Service Oriented Architecture orchestration. The purpose of the orchestration segment is to coordinate a group of other web-services. In the case of the prototype system, BPEL takes the values from the initial services, obtains the results, and pushes them into the fusion service. The final result is passed back to the BPEL module and the process completes. In this section we will examine two segments of processing logic, first is the BPEL process XML schema or XSD file, the second is the BPEL XML file. Note the input element matches the required input variables for both the SIGACT and Reconstruction Project EJB's.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xml.netbeans.org/schema/newXMLSchema"
        xmlns:tns="http://xml.netbeans.org/schema/newXMLSchema"
        elementFormDefault="qualified">
   <xsd:complexType name="processGeoTrackFusion">
      <xsd:sequence>
         <xsd:element name="SigactStart" nillable="true" type="xsd:string"/>
         <xsd:element name="SigactFinish" nillable="true" type="xsd:string"/>
         <xsd:element name="Sector" nillable="true" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="processGeoTrackFusionResponse">
      <xsd:sequence>
         <xsd:element name="return" nillable="true" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="processGeoTrackFusionResponse"
type="tns:processGeoTrackFusionResponse"/>
   <xsd:element name="processGeoTrackFusion"
type="tns:processGeoTrackFusion"/>
</xsd:schema>.
```

The next code snippet is the BPEL execution file. BPEL employs a number of standard functions, however the prototype employs invoke, assign, flow start and receive only. Netbeans 5.5 provides a graphical BPEL designer that allows the developer to rapidly create the process and map the elements. In the section below, note the declaration of the partner links. Each of the EJB's employed in the "GeoTrackFusion" process.

```
<?xml version="1.0" encoding="UTF-8"?>
<process
   name="GeoTrackFusion"
   targetNamespace="http://enterprise.netbeans.org/bpel/GeoTrackFusion"
   xmlns="http://schemas.xmlsoap.org/ws/2004/03/business-process/"
   ….
<partnerLinks>
      <partnerLink name="NewTrackEJB"
partnerLinkType="ns5:NewTrackServiceLinkType"
partnerRole="NewTrackServiceRole"/>
      <partnerLink name="GeoFuserEJB"
partnerLinkType="ns4:GeoTrackFusionLinkType"
partnerRole="GeoTrackFusionRole"/>
      <partnerLink name="ReconEJB"
partnerLinkType="ns3:ReconProjectEJBLinkType"
partnerRole="ReconProjectEJBRole"/>
      <partnerLink name="SigactEJB"
partnerLinkType="ns2:BaseSigactEJBLinkType"
```

The next section of the BPEL file highlights the initial assign functions.

```
  <sequence>
      <receive name="ReceiveFromBpel" createInstance="yes"
partnerLink="GeoTrackFusionBpel" operation="GeoTrackFusionOperation"
portType="ns1:GeoTrackFusionPortType"
variable="GeoTrackFusionOperationInput"/>
      <assign name="Assign1">
         <copy>

<from>$GeoTrackFusionOperationInput.Fusionmessage/ns0:SigactStart</from>
            <to>$ProcessBaseSIGACTEJBInput.parameters/arg0</to>
         </copy>
         <copy>

<from>$GeoTrackFusionOperationInput.Fusionmessage/ns0:SigactFinish</from>
            <to>$ProcessBaseSIGACTEJBInput.parameters/arg1</to>
         </copy>
         <copy>
            <from>$GeoTrackFusionOperationInput.Fusionmessage/ns0:Sector</from>
            <to>$GetReconProjectInput.parameters/Sector</to>
```

BPEL permits reuse of services in any number of processes. In the case of the prototype, we employ a simple process, however the ability to mix and match new services and fusion engines is the potential gains associated SOA and orchestration.

D. FUSION DISCOVERY PROTOTYPE

One of the essential elements of a SOA is the concept of service discovery. Discovery is the method by which the existence and location of a service resides.

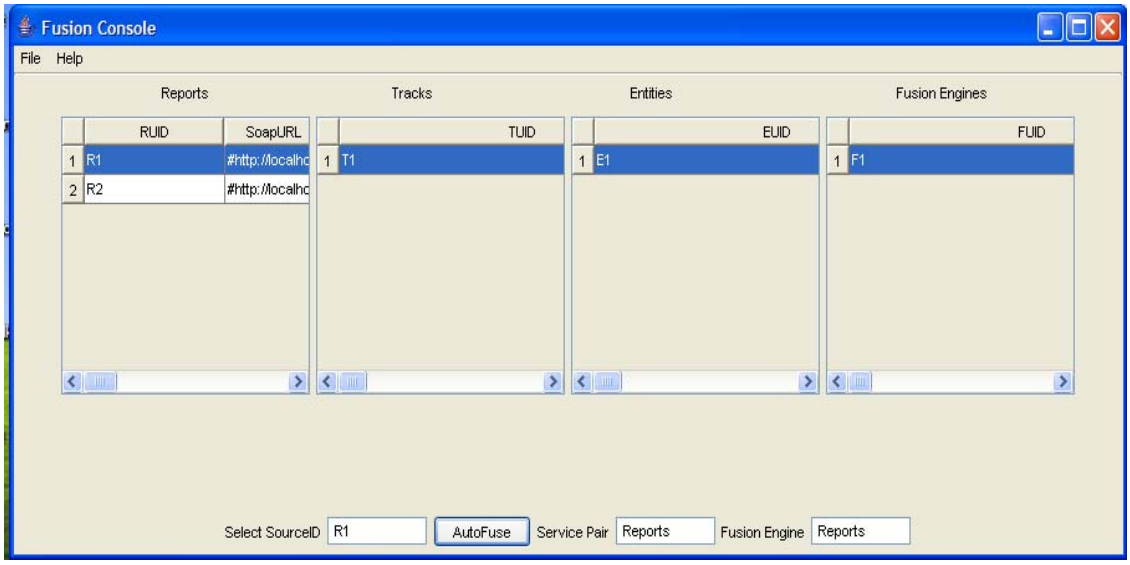

Figure 45. Auto-Fusion Discovery Application

As discussed in previous sections, auto-fusion provides a motivation for investment in a SOA. Integrating the previously discussed Auto-Fusion *Figure of Merit* into a discovery application allows a user or system to determine which fusion pairs and algorithms make sense. The source code below is an XML configuration file with pre-set values that when combined with the meta-data provided by the services allows the computation of the figure of merit. The prototypes goal is to assess what meta-data is required vice validating if a UDDI supports the role. Further research into UDDI is required to determine if a UDDI T-model notion supports the type of meta-data required to determine an auto-fusion FOM.

```
<?xml version="1.0"?>
<!--Configuration file for AutoFusion prototype-->
<FusionConfig>
        <vectorOne>
                <!--Desired computational complexity: low=2, med=5, high=10-->
          <computationalComplex>low</computationalComplex>
        </vectorOne>
        <vectorTwo>
                <!--Desired computational complexity:
limited=11KBPS,moderate=1mbps, LAN 11MBPS-->
          <networkComplex>limited</networkComplex>
        </vectorTwo>
        <vectorThree>
                <!--Desired Joint Director of Labs target fusion outcome
level###(One,Two,Three,Four)-->
          <JDLpreferred>levelTwo</JDLpreferred>
        </vectorThree>
</FusionConfig>
```

E. HOW THE PROTOTYPE VALIDATES THE THREE TIER MODEL

The goal of the *trickle-up* pattern is to separate the processing logic of the data object from the fusion logic, to support system agility. Present system architectures hardwire fusion logic to the data source or even conduct fusion logic directly on the database. This strongly coupled architecture results in significant difficulty in changing fusion algorithms and does not support a SOA environment. In the prototype each service operates independently and is only chained when BPEL drives a fusion process. This validates the potential viability of the architecture, however limitations remain using web-enabled SOA in fusion systems.

F. LIMITATIONS OF THE PROTOTYPE SOFTWARE AND RECOMMENDATION FOR FURTHER VALIDATION

The prototype demonstrates a separation of processing along SOA lines and advances the ability to plug and play different data sources and fusion engines. Although promising, a number of limitations exist. The central limitation stems from the need for a

common XML data object definition. For example, each service has an independent XML schema that drives a WSDL definition and the BPEL has its own too, which supports the orchestration, but to truly employ services without recompiling code requires a shared schema. Additionally, a number of fusion actions are time of receipt based, and employ embedded processors in a deterministic fashion. Porting those algorithms to a SOA web-service could prove difficult, especially in weapons grade, real-time systems.

IX. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

In the course of the research, I made a few observations as they relate to applying SOA technologies to DoD system challenges. These observations include a general migration of capabilities to internet technologies, a change in market leadership in the technical IT field and approaches for transitioning SOA technologies from laboratory to warfighter. In this final section we will first review the results of the research, discuss some of the observations described above and close with recommendations for future research.

The research started with a challenge of increasing volume of data and info and a potentially dramatic increase from exposing data using SOA technologies. To manage the data in a control and monitoring domain the research proposed use of two software patterns and a data fusion process. Using Gamma's standard design pattern definitions the research detailed the zone and trickle-up patterns. Next the research discussed the application of the patterns in a command and control challenge associated with the Navy's Maritime Domain Awareness system, and developed a prototype software system using SOA technologies to validate the patterns and process. Specifically the research validated the following:

- On the SOA data layer or infrastructure side we demonstrated the viability of the ***trickle-up*** pattern for a simple fusion problem, showing how we can divide the processing logic of the service from the fusion engine. At this phase you can readily see the need for a standard ontology and data schema. This type of semi-dynamic binding would be impossible without extensive knowledge of the meta-data. A common "track" definition is critical to this type of architecture.
- For control and monitoring software systems we detailed the *Zone* pattern, which discusses a distributed approach to sensor data management that employs the content developed by the *trickle-up* pattern and departs from both the user defined operational picture (UDOP) approach of all sensor data

migrating into a core or the traditional “all data push” employed by the current Joint Global Command and Control System.

- Research prototype demonstrated use of SOA web-service based orchestration (BPEL) to administer a data fusion process.
- Research prototype demonstrated an Auto-Fusion discovery method along with a *figure of merit* method based on network costs, computational costs and potential fusion “benefit” based on the four level fusion model and employing the *trickle-up* pattern.
- Research explored impacts of an auto-fusion requirement as a broad Data Strategy goal for the Maritime Domain Awareness system.

In the following section we will discuss some general observations and recommendations from the research and close with some recommendations for future research.

In 1989 the Defense Communication Agency (now DISA) established the internet as a commercial activity and split the non-military segments of ARPANET to a consortium of government, industry and academic leaders (Hobbes', 2). Early adopters quickly realized the potential of the network, however it still took some years for applications to gain wide acceptance. Some of the hurdles stemmed from the complexity of the PC user interface as compared with a telephone or television set, while other hurdles stemmed from cost and availability of applications on the internet. One of the methods to spur adoption of new technologies was the availability of shareware or freeware products. For example, early adopters to the internet found little compelling content or applications, however a freeware product known as Mosaic (a freeware HTML browser), opened hyperextend images and graphics to a wider array of users, and put a “friendly” face on the Internet’s content. Mosaic put a “face” on the internet lowering the technological barrier of entry of users into the content of the internet. In a similar fashion, Service Oriented Architectures for DoD applications too require a “face” and more importantly a capability. DoD sponsored SOA demonstrations such as Horizontal Fusion (OSD), Net Centric Capabilities Pilot (DISA) and Sea-Trial (ONR) provide venues to put an operational face of SOA’s and have done so with reasonable success, despite the difficulties of articulating benefits whose real potential do not manifest themselves until a

critical mass occurs and systems are widely adopted. The zone and trickle-up pattern provide a "face" to a control and monitoring domain software, which articulates to developers the three relationships a C2 monitoring system must enable and the sources they need. This "face" of SOA is a data management tool which manages the dynamic discovery and addition of content at runtime of data sources operating on a *trickle-up* SOA, such as TADIL, ELINT, SIGINT, raw and processed intelligence. This face or "Kmart COP" is the "last mile" managing the complexity of the architecture. To deliver the critical mass desired a Kmart COP must ride on a suitable SOA that too is non-proprietary in its binding between clients and data sources as discussed earlier in the research. Additionally, the Kmart COP tool set must extend the authority and user experience of collaborative systems to managing an approved common operations picture as discussed in MDA CONOPS section of this paper. To gain initial user acceptance the tool must improve some near term user requirement, allowing a gradual adoption of the additional data richness and functionality. For example, when Microsoft first marketed the Outlook email program, users did not utilize much of the calendar and tasking tools provided. However over time, these tools gained wide acceptance and drove users to demand the increased functionality associated with using a Microsoft exchange server. In a similar fashion, Kmart COP, must provide an initial capability of access to present data, display it and allow the creation of a owned COP that can be collaborated on in and integrated application. Specifically, the tool must:

- Provide for the dynamic discovery of SOA track sources (*trickle-up).*
- Provide capability to add and remove data sources at run time (*trickle-up).*
- Provide integrated collaborative software between the data and the visualization segment (*zone* pattern).
- Provide a mapping/visualization segment.
- Provide mechanism to dynamically subscribe to other Kmart COP's and integrate their managed pictures into a common picture (*zone* pattern)

- Leverage the capability of SOA segments to uniquely identify messages to provide a unique identification tag to mitigate data ringing (duplicative tracks displayed for the same contact).
- Conduct the data-management in a method to efficiently utilize bandwidth to support a disconnected or limited bandwidth user base.

In the preceding section we discussed observations relating to taking the *zone* and *trickle-up* pattern inspired software and migrating them from laboratory to warfighter, the next section discusses how the DoD is no longer the leader in information technology and needs to adopt from leader to participant in information technology standard development as well as recommendations of how acquisition managers can approach SOA technologies.

Post World War II, defense contractors led industry in the development of high tech communications and computing systems. The defense industries lead began to fade during the 1980's as personnel electronics and computing systems developed for commercial use proliferated. Presently commercial industry leads many of the high tech factors and hence drives many of the commercial standards. Utilization of these commercial standards significantly saves the DoD resources by shifting development costs to industry. Since the DoD is a large consumer of industry standards based systems it must maintain active participation in standard committee's and development boards.

An additional requirement beyond interoperability standard development is the development of non-proprietary systems. These non-proprietary systems can maintain interoperability and extensibility while employing the incentives of industry development. The Net-Centric Enterprise Solutions for Interoperability (NESI) method provides critical contracting language and references to develop non-proprietary military systems (NESI, Vol 1). Beyond NESI, initial core systems should be developed with strong government oversight. A potential solution is to focus initial development on multi-contractor supported, government led development in government labs such as SPAWAR System Centers. Only with tight government oversight can a product without hidden hooks be developed and initially deployed. That is not to say, SOA's have no

place for proprietary products and segments. In reality SOA's can foster small businesses participation by lowering integration costs and increased sharing of solutions that an open SOA provides. The first brave steps however, must be made by agents of the government.

A great deal of written material is available discussing the shortfalls of present Command and Control systems, and extolling the virtues of a net-centric and service-oriented architectures (SOA). Books such as Power to the Edge and Net-Centric Warfare detail many of the why's of SOA (Alberts, 15), this research rather focused on answering some of the how's. Overall the message from the research is to demonstrate that although in the past, technology and bandwidth limited the potential for a SOA or web-based architecture for DoD solutions; the present state of the technology is in-fact mature enough to start fielding real systems that employ some of the technologies provided by web-enabled SOA's.

Service Oriented Architectures offer a compelling solution to a number of the Navy's command and control data management needs. A significant departure from existing technologies and methods, there are both acquisition and warfighter doctrinal impacts. A change in tool utilization alone will not deliver the results the DoD requires, doctrine must be altered concurrently. Additionally, some of SOA's potential gains require a level of openness that runs contrary to the culture of defense contractors. This is not to say that all applications need to be non-proprietary/open source, just that the core communications between the services and specifications need to remain non-proprietary and vendor neutral to maximize open communications between systems and realize the transformation in Command and Control required to make FORCEnet a reality.

The dilemma acquisition managers have today remain unchanged since sword makers in Athens toiled with metallurgy; balancing the risks of new technologies, in a resource constrained environment against the demands to maintain present capabilities. Program offices answer to a chorus of various entities to become more "transformational" while functioning in the real world of maintaining and upgrading an existing system, and fielding systems in an extremely challenging fiscal environment. Despite, these obstacles

however, we need to take the next steps toward a net-centric vision and address the real and pressing poor record of both a technology refresh and expanded capabilities in the C2 field.

A. RECOMMENDATIONS FOR FUTURE RESEARCH

Numerous future research topics exist in the SOA domain, especially as it relates to using wed-service DoD technologies to address interoperability challenges. The use of web-based SOA framework for tactical uses in particular, creates a series of challenges relating to adopting what is essentially a business oriented solution to the rigors of a combat requirement. Quality of service topics such as reliability, security and maintainability require evaluation as well as topics to determine how well SOA based systems perform in a netcentric environment where high latency and low bandwidth are common. The following are a list of potential topics:

SOA Discovery Methods

In this research the prototype discovery method was not build on an industry standard. A future research endeavor would include adapting *the auto-fusion figure of merit* to a web service registry standard such as Universal Discovery Directory Index (UDDI) or enterprise business extensible markup language (ebXML) methods. Interesting challenges surround adopting the registry descriptions of the *trickle-up* pattern of *reports, tracks* and *entities* to the UDDI T-model structure. Additionally, since the UDDI standard does not define data elements to support a rules based determination of service pair selection, interesting research potentially can focus on where in a SOA framework should the logic reside to determine which services are potential fusion partners.

SOA Eventing Methods

In this research the prototype employed stateless web-service SOA architecture to support the research goals. Potentially future research can examine using an Enterprise Service Bus (ESB), Java Messaging Service (JMS) or Web Service (*WS) Eventing based SOA architectures to support a more dynamic environment. Adapting the Business Process Execution Language for web services (BPEL4WS) to operate on a set of JMS topics for a defined time period would be an interesting challenge. Additional topics

along the same lines could include network performance challenges, data in transit security issues as well as the challenge in adapting the BPEL engine to orchestrate a continuous process.

Data Models and Ontology's

The research presented in this paper touched on data schemas and models, but the prototype software employed only simple XML data definitions of a globally unique identification, name and position. The DoD meta-data registry has a number of Command and Control or Intelligence Surveillance and Reconnaissance data definitions and schemas to employ. As discussed previously a standard data definition is critical for the type of late-binding advocated by the research to support auto-fusion.

SOA Framework

Frank Bushmann in his text *Pattern Oriented Software Architecture* defines an application framework as an "integrated set of components that collaborate to provide reusable software architecture for a family of related applications" (Bushmann, 554).

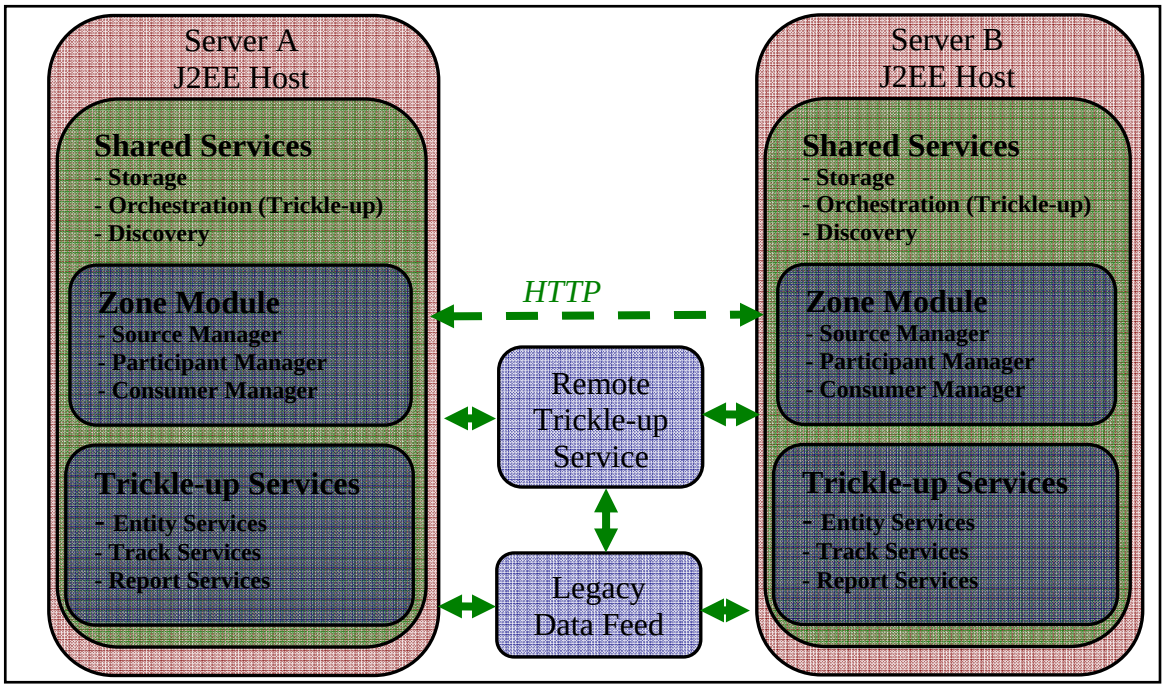

Figure 46. Pattern and Process Research Framework

The same concept can be applied to a SOA framework discussed in this dissertation. The trickle-up and zone pattern, supported by a web service SOA framework and executing the auto-fusion process provides an interesting control and monitoring software framework.

The physical manifestation of an auto-fusion framework is a collection of services on a system platform with interoperability maintained by open standard web-service transports. Figure 46 shows a reference implementation of the two patterns and process discussed in the dissertation. Additionally, figure 46 shows two separate servers, with a J2EE compliant web-hosting environment. The J2EE environment provides web-hosting, discovery, and orchestration services, for both the zone and trickle-up pattern services. The discovery service registers the pattern services and coupled with the orchestration services executes the auto-fusion process detailed in earlier sections. Furthermore, figure 46 shows a remote trickle-up service as well as a legacy data feed inject. The difference between the remote trickle-up and the trickle-up module in the servers in the diagram is the absence of the shared services located on the server. Specifically, the remote trickle up does not require orchestration and discovery. The remote trickle-up server does however require persistence and web-hosting of some kind, but a full suite of core or shared services is not required, hence limiting the software footprint of a trickle-up service. Another item shown in the diagram is a legacy data feed. These legacy feeds are critical to a migration from a non-SOA based system to a SOA framework.

A number of future research topics surround the development of a framework from the two patterns and process detailed in the research. These research topics include how to federate discovery services between servers, and what data transport methods best synchronize services. Additionally, different orchestration methods should be examined as well as choreography (a less centralized form of orchestration).

Trickle-up and Zone Pattern in DoD Architecture Framework

The Department of Defense Architecture Framework (DoDAF) “enables architecture descriptions to be compared and related across organizational boundaries, including Joint and multinational boundaries “(DoDAF Deskguide, 1-1). DoDAF is a

common description language for DoD acquisitions and is comprised of three general views of a system; operational view (OV), system view (SV), and technical view (TV). Although this dissertation provides SOA design and UML diagrams to aid understanding of the trickle-up and zone pattern, developing DoDAF artifacts of the patterns could further assist in application of the patterns. Specifically, system and technical views can further describe the interactions between the two patterns and outside systems and an operational view can aid in understanding the impact of the patterns on C4I system capabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. SOURCE CODE

```
/*
 * BaseSigactEJB.java
 *
 * Started on April 18, 2007, 12:15 AM
 * LCDR Kurt Rothenhaus
 * Created in support of Dissertation Research
 * Service provides SOA based method to access
 * Iraq Reconstruction Project data based on sector of
 * economy (Water, Electric, Health, Justice)
 */

package com.sun.BaseSigactEJB;

import java.util.ArrayList;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebParam.Mode;
import javax.jws.WebService;
import com.borland.dx.sql.dataset.*;
import com.borland.dbswing.*;
import java.sql.Timestamp;
import java.util.List;
import javax.xml.ws.Holder;

/**
 *
 * @author kjrothen
 */

@Stateless()
@WebService()
public class BaseSigactEJB {

    Database database1 = new Database();
    QueryDataSet queryDataSet1 = new QueryDataSet();

    boolean stat = false;

    /**
    * Web service operation
    */
```

```
    @WebMethod
    public List<sigact> processBaseSIGACTEJB(String sigactStart, String sigactFinish) {

        return getSigBatch(sigactStart,sigactFinish);
    }


    public List<sigact> getSigBatch(String sigactStart, String sigactFinish){


       String start = sigactStart;
       String finish = sigactFinish;


     queryDataSet1.close();

     queryDataSet1.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(database1,
"SELECT
SIGACT.EventType,SIGACT.EventCategory,SIGACT.OccuredWhen,SIGACT.PostedW
hen,SIGACT.region,SIGAC" +

"T.MGRS,SIGACT.PrimaryTarget,SIGACT.TargetCategory,SIGACT.City,SIGACT.Pro
vince " +
       "FROM \"C:\\Documents and Settings\\kjrothen\\My
Documents\\SIGACT\".SIGACT"
       + " WHERE OccuredWhen BETWEEN #" + start + "# AND #" + finish + "# ", null,
true, Load.ALL));

      database1.setConnection(new
com.borland.dx.sql.dataset.ConnectionDescriptor("jdbc:odbc:SIGACT", "", "", false,
"sun.jdbc.odbc.JdbcOdbcDriver"));
      database1.setUseCaseSensitiveId(true);
      database1.setUseSpacePadding(false);
      database1.setDatabaseName("");
      queryDataSet1.open();

      String EventType = "begin";
      String EventCategory = "null";
      String region = "null";
      Timestamp OccuredWhen = null;
      String MGRS = "null";
      String PrimaryTarget = "null";
      String TargetCategory = "null";
      String City = "null";
```

```
String Province = "null";

 List<sigact> sigie = new ArrayList<sigact>();

 while (queryDataSet1.inBounds()) {

     EventType = queryDataSet1.getString("EventType");
     EventCategory = queryDataSet1.getString("EventCategory");
     OccuredWhen = queryDataSet1.getTimestamp("OccuredWhen");

     double userlon = 1.1;
     double userlat = 2.1; //queryDataSet1.getDouble("Lat");
     region = queryDataSet1.getString("region");
     MGRS = queryDataSet1.getString("MGRS");
     String MGR = MGRS.replaceAll(" ", "");
     Position R = MGRStoLat(MGR);

      userlat = R.getLat();
      userlon = R.getLng();

     PrimaryTarget = queryDataSet1.getString("PrimaryTarget");
     TargetCategory = queryDataSet1.getString("TargetCategory");
     City = queryDataSet1.getString("City");
     Province = queryDataSet1.getString("Province");

     System.out.println(PrimaryTarget);
     System.out.println(TargetCategory);


      sigact sig = new sigact();
      sig.setAttackTarget(PrimaryTarget);
      sig.setSigLat(userlat);
      sig.setSigLon(userlon);
      sigie.add(sig);

    queryDataSet1.next();


 }  //End While Loop
    queryDataSet1.close();
    return sigie;


 }
```

```
public Position MGRStoLat(String Mgrs){


    UTM utm = new UTM();

   try {
    MGR.MRGtoUTM(Mgrs, utm);
    System.out.println("MRG=" + Mgrs);
    System.out.println("UTM=" + utm);
    Position p = utm.toPosition();
    System.out.println(" LL=" + p);
    double lat = p.getLat();
    double lng = p.getLng();
    return p;
    }
    catch (Exception e) {
      MGR.MRGtoUTM("01NGD1200015000", utm);
      Position p = utm.toPosition();
      System.out.println("Catch the exception =" + p);
      return p;
      }
    }


}

package com.sun.BaseSigactEJB;

public class Ellipsoid {
  //
  public int m_id;
  public String m_ellipsoidName;
  public double m_EquatorialRadius;
  public double m_eccentricitySquared;

  //
  public Ellipsoid(int id, String ellipsoidName, double EquatorialRadius,
                double eccentricitySquared) {
    m_id = id;
    m_ellipsoidName = ellipsoidName;
    m_EquatorialRadius = EquatorialRadius;
    m_eccentricitySquared = eccentricitySquared;
  }
```

```
  //
  static public Ellipsoid m_ElipseTable[] = {
     new Ellipsoid(0, "Placeholder", 0.0, 0.0),
     new Ellipsoid(1, "Airy", 6377563.0, 0.00667054),
     new Ellipsoid(2, "Australian National", 6378160.0, 0.006694542),
     new Ellipsoid(3, "Bessel 1841", 6377397.0, 0.006674372),
     new Ellipsoid(4, "Bessel 1841 (Nambia) ", 6377484.0, 0.006674372),
     new Ellipsoid(5, "Clarke 1866", 6378206.0, 0.006768658),
     new Ellipsoid(6, "Clarke 1880", 6378249.0, 0.006803511),
     new Ellipsoid(7, "Everest", 6377276.0, 0.006637847),
     new Ellipsoid(8, "Fischer 1960 (Mercury) ", 6378166.0, 0.006693422),
     new Ellipsoid(9, "Fischer 1968", 6378150.0, 0.006693422),
     new Ellipsoid(10, "GRS 1967", 6378160.0, 0.006694605),
     new Ellipsoid(11, "GRS 1980", 6378137.0, 0.00669438),
     new Ellipsoid(12, "Helmert 1906", 6378200.0, 0.006693422),
     new Ellipsoid(13, "Hough", 6378270.0, 0.00672267),
     new Ellipsoid(14, "International", 6378388.0, 0.00672267),
     new Ellipsoid(15, "Krassovsky", 6378245.0, 0.006693422),
     new Ellipsoid(16, "Modified Airy", 6377340.0, 0.00667054),
     new Ellipsoid(17, "Modified Everest", 6377304.0, 0.006637847),
     new Ellipsoid(18, "Modified Fischer 1960", 6378155.0, 0.006693422),
     new Ellipsoid(19, "South American 1969", 6378160.0, 0.006694542),
     new Ellipsoid(20, "WGS 60", 6378165.0, 0.006693422),
     new Ellipsoid(21, "WGS 66", 6378145.0, 0.006694542),
     new Ellipsoid(22, "WGS-72", 6378135.0, 0.006694318),
     new Ellipsoid(23, "WGS-84", 6378137.0, 0.00669438)
  };
};

package com.sun.BaseSigactEJB;


public class Location {

  public final static double DEF_DOUBLE = -32768.0;
  public static double DEFAULT_LAT = DEF_DOUBLE;
  public static double DEFAULT_LNG = DEF_DOUBLE;

  private String m_location;

  public Location() {
    m_location = "";
  }

  public Location(String str) {
```

```
    m_location = str;
  }

  static public Position StringToPosition(String pos) {
    Location location = new Location(pos);
    Position position = new Position();
    if (location.getPosition(position)) {
      return (position);
    }
    return (null);
  }

  /**
   * Set the Location string.  The Location of a track can be given in a
   * variety of formats including any of the following:
   *
   * Lat/Lng degrees (DDA DDDA)
   * Lat/Lng minutes (DDMMA DDDMMA,LM:DDMMADDDMMA)
   * Lat/Lng seconds (DDMMSSA DDDMMSSA,LS:DDMMSSADDDMMSSA)
   * Lat/Lng decimal seconds (DDMMSS.SA DDDMMSS.SA,DD:MM:SS.SA
DDD:MM:SS.SA)
   * Lat/Lng decimal hundredths (DDMMSS.SSA DDDMMSS.SSA,DD:MM:SS.SSA
DDD:MM:SS.SSA)
   *
   * Lat/Lng like:
   * LL:DDMMSSNc-SSSMMSSWc
   * LL:DDMMSS.SNc-SSSMMSS.SWc
   * LL:DDMMSS.SSNc-SSSMMSS.SSWc
   * LL:DDMMNc-SSSMMWc
   * LL:DDMM.MMMNc-SSSMM.MMMWc
   *
   * UT:31NAD1200015000 Zone=31, Lat Band=N, 100000 cell=A+D, Easting=12000,
Northing=15000
   * MGR
   * Georef (AAAANNNN)
   *
   * And if all else fails we try Lat,Lng in the form +12.12345 -123.20000 (i.e. two
doubles);
   *
   * @param str Location string.
   */
  public void setLocation(String str) {
    m_location = str;
  }
```

```
/**
 * Get the Location string.
 * @return Location string.
 */
public String getLocation() {
  return (m_location);
}

public Position getPosition() {
  Position pos = new Position();
  if (getPosition(pos)) {
    return (pos);
  }
  return (null);
}

/**
 * Fill an object using its IxPosition interface
 * @param posit
 */
public boolean getPosition(Position posit) {
  String latStr;
  String lngStr;
  if (patternMatch("####N#####W")) {
    latStr = m_location.substring(0, 5);
    lngStr = m_location.substring(5);
    posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
    return (true);
  }
  if (patternMatch("######N#######W")) {
    latStr = m_location.substring(0, 7);
    lngStr = m_location.substring(7);
    posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
    return (true);
  }
  if (patternMatch("LD:##N###W")) {
    latStr = m_location.substring(3, 6);
    lngStr = m_location.substring(6, 10);
    posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
    return (true);
  }
  if (patternMatch("LM:####N#####W")) {
    latStr = m_location.substring(3, 8);
    lngStr = m_location.substring(8, 14);
    posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
```

```
  return (true);
 }
 if (patternMatch("LS:######N#######W")) {
  latStr = m_location.substring(3, 10);
  lngStr = m_location.substring(10, 18);
  posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
  return (true);
 }
 if (patternMatch("LS:######.#N#######.#W")) {
  latStr = m_location.substring(3, 12);
  lngStr = m_location.substring(12, 22);
  posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
  return (true);
 }
 if (patternMatch("LC:######.#N#######.#W")) {
  latStr = m_location.substring(3, 12);
  lngStr = m_location.substring(12, 22);
  posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
  return (true);
 }
 if (patternMatch("LL:####N#-#####W#")) {
  latStr = m_location.substring(3, 8);
  lngStr = m_location.substring(10, 16);
  posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
  return (true);
 }
 if (patternMatch("LL:####.###N#-#####.###W#")) {
  latStr = m_location.substring(3, 12);
  lngStr = m_location.substring(14, 24);
  posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
  return (true);
 }
 if (patternMatch("LL:######N#-#######W#")) {
  latStr = m_location.substring(3, 10);
  lngStr = m_location.substring(12, 20);
  posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
  return (true);
 }
 if (patternMatch("LL:######.#N#-#######.#W#")) {
  latStr = m_location.substring(3, 12);
  lngStr = m_location.substring(14, 24);
  posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
  return (true);
 }
 if (patternMatch("LL:######.##N#-#######.##W#")) {
```

```
    latStr = m_location.substring(3, 13);
    lngStr = m_location.substring(15, 26);
    posit.setLatLng(StringToLat(latStr), StringToLng(lngStr));
    return (true);
   }
   // 1 & 10-Meter MGRS
   if (m_location.startsWith("UT:") || m_location.startsWith("UH:")) {
    UTM utm = new UTM();
    if (MGR.MRGtoUTM(m_location.substring(3), utm)) {
     return (UTM.UTMtoLL(utm, posit));
    }
   }
   // 1-1000 Meter MGRS
   if (isMGR()) {
    UTM utm = new UTM();
    if (MGR.MRGtoUTM(m_location, utm)) {
     return (UTM.UTMtoLL(utm, posit));
    }
   }
   //
   String parts[] = m_location.trim().split(" ");
   if (parts.length != 2) {
    parts = m_location.trim().split("-");
    if (parts.length != 2) {
     return (false);
    }
   }
   //
   latStr = parts[0];
   lngStr = parts[1];
   double lat = StringToLat(latStr);
   double lng = StringToLng(lngStr);
   if (lat == DEFAULT_LAT || lng == DEFAULT_LNG) {
    return (false);
   }
   posit.setLatLng(lat, lng);
   return (true);
  }

  public boolean isGEOREF() {

   if (m_location.length() != 8) {
    return (false);
   }
   int i;
```

```
  for (i = 0; i < 4; i++) {
    if (StringOps.validAlpha.indexOf(m_location.substring(i, i + 1)) == -1) {
      return (false);
    }
  }
  for (i = 4; i < 8; i++) {
    if (StringOps.validDigit.indexOf(m_location.substring(i, i + 1)) == -1) {
      return (false);
    }
  }
  return (true);
}

public boolean isMGR() {
  if (patternMatch("##$$$##########")) {
    return (true);
  }
  if (patternMatch("##$$$########")) {
    return (true);
  }
  if (patternMatch("##$$$######")) {
    return (true);
  }
  if (patternMatch("##$$$####")) {
    return (true);
  }
  return (false);
}

public boolean isUTM() {
  return (patternMatch("## ###### ########"));
}

public boolean isLatLng() {
  if (patternMatch("##N ###W")) {
    return (true);
  }
  if (patternMatch("####N #####W")) {
    return (true);
  }
  if (patternMatch("####N#####W")) {
    return (true);
  }
  if (patternMatch("######N #######W")) {
    return (true);
```

```
  }
  if (patternMatch("######N#######W")) {
    return (true);
  }
  if (patternMatch("######.#N #######.#W")) {
    return (true);
  }
  if (patternMatch("##:##N ###:##W")) {
    return (true);
  }
  if (patternMatch("##:##:##N ###:##:##W")) {
    return (true);
  }
  if (patternMatch("##:##:##.#N ###:##:##.#W")) {
    return (true);
  }
  if (patternMatch("LD:##N###W")) {
    return (true);
  }
  if (patternMatch("LM:####N#####W")) {
    return (true);
  }
  if (patternMatch("LS:######N#######W")) {
    return (true);
  }
  if (patternMatch("LS:######.#N#######.#W")) {
    return (true);
  }
  return (false);
}

public boolean patternMatch(String pattern) {
  return (StringOps.patternMatch(pattern, m_location));
}

public static double StringToLat(String str) {
  //
  int N = str.indexOf("N");
  int S = str.indexOf("S");
  //
  if (N == -1 && S == -1) {
    // Test for floating point between -90 and 90
    double lat = DEFAULT_LAT;
    try {
      lat = Double.parseDouble(str);
```

```
  }
  catch (Exception e) {}
  if (lat > 90.0) {
    return (DEFAULT_LAT);
  }
  if (lat < -90.0) {
    return (DEFAULT_LAT);
  }
  return (lat);
}
int n = Math.max(N, S);
str = str.substring(0, n + 1);
//
String sgnStr = "0";
String degStr = "0";
String minStr = "0";
String secStr = "0";
//
if (StringOps.patternMatch("##N", str)) {
  degStr = str.substring(0, 2);
  minStr = "0";
  secStr = "0";
  sgnStr = str.substring(2, 3);
}
else if (StringOps.patternMatch("####N", str)) {
  degStr = str.substring(0, 2);
  minStr = str.substring(2, 4);
  secStr = "0";
  sgnStr = str.substring(4, 5);
}
else if (StringOps.patternMatch("####.###N", str)) {
  degStr = str.substring(0, 2);
  minStr = str.substring(2, 8);
  secStr = "0";
  sgnStr = str.substring(8, 9);
}
else if (StringOps.patternMatch("######N", str)) {
  degStr = str.substring(0, 2);
  minStr = str.substring(2, 4);
  secStr = str.substring(4, 6);
  sgnStr = str.substring(6, 7);
}
else if (StringOps.patternMatch("######.#N", str)) {
  degStr = str.substring(0, 2);
  minStr = str.substring(2, 4);
```

```
    secStr = str.substring(4, 8);
    sgnStr = str.substring(8, 9);
   }
   else if (StringOps.patternMatch("######.##N", str)) {
    degStr = str.substring(0, 2);
    minStr = str.substring(2, 4);
    secStr = str.substring(4, 9);
    sgnStr = str.substring(9, 10);
   }
   else if (StringOps.patternMatch("##:##:##N", str)) {
    degStr = str.substring(0, 2);
    minStr = str.substring(3, 5);
    secStr = str.substring(6, 8);
    sgnStr = str.substring(8, 9);
   }
   else if (StringOps.patternMatch("##:##:##.#N", str)) {
    degStr = str.substring(0, 2);
    minStr = str.substring(3, 5);
    secStr = str.substring(6, 10);
    sgnStr = str.substring(10, 11);
   }
   else {
    return (DEF_DOUBLE);
   }

   Double deg = new Double(degStr);
   Double min = new Double(minStr);
   Double sec = new Double(secStr);
   if (sgnStr.equals("N")) {
    return ( (deg.doubleValue() +
            (min.doubleValue() + sec.doubleValue() / 60) / 60));
   }
   else if (sgnStr.equals("S")) {
    return ( - (deg.doubleValue() +
              (min.doubleValue() + sec.doubleValue() / 60) / 60));
   }
   else {
    return (DEFAULT_LAT);
   }
  }

  public static double StringToLng(String str) {
   //
   int E = str.indexOf("E");
   int W = str.indexOf("W");
```

```
//
if (E == -1 && W == -1) {
  // Test for floating point between -180 and 180
  double lng = DEFAULT_LNG;
  try {
    lng = Double.parseDouble(str);
  }
  catch (Exception e) {
    return (DEFAULT_LNG);
  }
  if (lng > 180.0) {
    return (DEFAULT_LNG);
  }
  if (lng < -180.0) {
    return (DEFAULT_LNG);
  }
  return (lng);
}
int n = Math.max(E, W);
str = str.substring(0, n + 1);

String sgnStr = "0";
String degStr = "0";
String minStr = "0";
String secStr = "0";

if (StringOps.patternMatch("###W", str)) {
  degStr = str.substring(0, 3);
  minStr = "0";
  secStr = "0";
  sgnStr = str.substring(3, 4);
}
else if (StringOps.patternMatch("#####W", str)) {
  degStr = str.substring(0, 3);
  minStr = str.substring(3, 5);
  secStr = "0";
  sgnStr = str.substring(5, 6);
}
else if (StringOps.patternMatch("#####.###W", str)) {
  degStr = str.substring(0, 3);
  minStr = str.substring(3, 9);
  secStr = "0";
  sgnStr = str.substring(9, 10);
}
else if (StringOps.patternMatch("#######W", str)) {
```

```
  degStr = str.substring(0, 3);
  minStr = str.substring(3, 5);
  secStr = str.substring(5, 7);
  sgnStr = str.substring(7, 8);
}
else if (StringOps.patternMatch("#######.#W", str)) {
  degStr = str.substring(0, 3);
  minStr = str.substring(3, 5);
  secStr = str.substring(5, 9);
  sgnStr = str.substring(9, 10);
}
else if (StringOps.patternMatch("#######.##W", str)) {
  degStr = str.substring(0, 3);
  minStr = str.substring(3, 5);
  secStr = str.substring(5, 10);
  sgnStr = str.substring(10, 11);
}
else if (StringOps.patternMatch("###:##:##W", str)) {
  degStr = str.substring(0, 3);
  minStr = str.substring(4, 6);
  secStr = str.substring(7, 9);
  sgnStr = str.substring(9, 10);
}
else if (StringOps.patternMatch("###:##:##.#W", str)) {
  degStr = str.substring(0, 3);
  minStr = str.substring(4, 6);
  secStr = str.substring(7, 11);
  sgnStr = str.substring(11, 12);
}
else {
  return (DEFAULT_LNG);
}

Double deg = new Double(degStr);
Double min = new Double(minStr);
Double sec = new Double(secStr);
if (sgnStr.equals("E")) {
  return ( (deg.doubleValue() +
          (min.doubleValue() + sec.doubleValue() / 60) / 60));
}
else if (sgnStr.equals("W")) {
  return ( - (deg.doubleValue() +
            (min.doubleValue() + sec.doubleValue() / 60) / 60));
}
else {
```

```
    return (DEFAULT_LNG);
   }
  }

  public boolean isValid() {
   return (true);
  }
}

package com.sun.BaseSigactEJB;


import java.util.Hashtable;
import java.text.DecimalFormat;

public class MGR {
 static Hashtable  m_latitudeBandTable = new Hashtable();

  //
  class LatitudeBand {
   //
   String m_letter; /* letter representing latitude band  */
   double m_minNorthing; /* minimum northing for latitude band */
   double m_north; /* upper latitude for latitude band   */
   double m_south; /* lower latitude for latitude band   */

   //
   public LatitudeBand(String letter, double minNorthing, double north,
                       double south) {
     m_letter = letter;
     m_minNorthing = minNorthing;
     m_north = north;
     m_south = south;
     m_latitudeBandTable.put(m_letter, this);
   }
  };

  //
  static public LatitudeBand getBandByLetter(String letter) {
   return ( (LatitudeBand) m_latitudeBandTable.get(letter));
  }

  //
  LatitudeBand LatitudeBandTable[] = {
     new LatitudeBand("C", 1100000.0, -72.0, -80.5),
```

```
    new LatitudeBand("D", 2000000.0, -64.0, -72.0),
    new LatitudeBand("E", 2800000.0, -56.0, -64.0),
    new LatitudeBand("F", 3700000.0, -48.0, -56.0),
    new LatitudeBand("G", 4600000.0, -40.0, -48.0),
    new LatitudeBand("H", 5500000.0, -32.0, -40.0),
    new LatitudeBand("J", 6400000.0, -24.0, -32.0),
    new LatitudeBand("K", 7300000.0, -16.0, -24.0),
    new LatitudeBand("L", 8200000.0, -8.0, -16.0),
    new LatitudeBand("M", 9100000.0, 0.0, -8.0),
    new LatitudeBand("N", 0.0, 8.0, 0.0),
    new LatitudeBand("P", 800000.0, 16.0, 8.0),
    new LatitudeBand("Q", 1700000.0, 24.0, 16.0),
    new LatitudeBand("R", 2600000.0, 32.0, 24.0),
    new LatitudeBand("S", 3500000.0, 40.0, 32.0),
    new LatitudeBand("T", 4400000.0, 48.0, 40.0),
    new LatitudeBand("U", 5300000.0, 56.0, 48.0),
    new LatitudeBand("V", 6200000.0, 64.0, 56.0),
    new LatitudeBand("W", 7000000.0, 72.0, 64.0),
    new LatitudeBand("X", 7900000.0, 84.5, 72.0),
};
//
static String m_northernRows = "NPQRSTUVWX";
static String m_southernRows = "CDEFGHJKLM";
static String m_rowLetters = "ABCDEFGHJKLMNPQRSTUV";

//
int m_zone;
String m_row;
String m_cellRow;
String m_cellCol;
int m_easting;
int m_northing;
int m_precision;

//
public MGR() {}

//
public String getHemisphere() {
  if (m_northernRows.indexOf(m_row) != -1) {
    return ("N");
  }
  if (m_southernRows.indexOf(m_row) != -1) {
    return ("S");
  }
```

```
  return (null);
 }

 //
 public boolean setValue(String mgr) {
  if ( (mgr.length() != 15) && (mgr.length() != 13) && (mgr.length() != 11) &&
     (mgr.length() != 9) && (mgr.length() != 7)) {
   return (false);
  }
  try {
   m_zone = Integer.parseInt(mgr.substring(0, 2));
   m_row = mgr.substring(2, 3);
   m_cellCol = mgr.substring(3, 4);
   m_cellRow = mgr.substring(4, 5);
   String eastAndNorth = mgr.substring(5);
   int n = eastAndNorth.length() / 2;
   String zeros = "";
   if (n < 5) {
    zeros = StringOps.padString(zeros, "0", 5 - n);
   }
   m_easting = Integer.parseInt(mgr.substring(5, 5 + n) + zeros);
   m_northing = Integer.parseInt(mgr.substring(5 + n, 5 + 2 * n) + zeros);
   m_precision = n;
   //
   // These three zones in row 'X' don't exist
   //
   if (m_row.equals("X") &&
      ( (m_zone == 32) || (m_zone == 34) || (m_zone == 36))) {
    return (false);
   }

   return (true);
  }
  catch (Exception e) {
   return (false);
  }
 }

 //
public static boolean MRGtoUTM(String val, UTM utm) {
  MGR mgr = new MGR();
  mgr.setValue(val);
  String hemisphere = mgr.getHemisphere();
  String start = "A";
  String end = "H";
```

```
String sequence = "ABCDEFGH";
long falseNorthing = 0;
switch (mgr.m_zone % 6) {
  case 1:
    start = "A";
    end = "H";
    sequence = "ABCDEFGH";
    falseNorthing = 0;
    break;
  case 2:
    start = "J";
    end = "R";
    sequence = "JKLMNPQR";
    falseNorthing = 1500000;
    break;
  case 3:
    start = "S";
    end = "Z";
    sequence = "STUVWXYZ";
    falseNorthing = 0;
    break;
  case 4:
    start = "A";
    end = "H";
    sequence = "ABCDEFGH";
    falseNorthing = 1500000;
    break;
  case 5:
    start = "J";
    end = "R";
    sequence = "JKLMNPQR";
    falseNorthing = 0;
    break;
  case 0:
    start = "S";
    end = "Z";
    sequence = "STUVWXYZ";
    falseNorthing = 1500000;
    break;
}
if (mgr.m_cellCol.compareTo(start) < 0) {
  return (false);
}
if (mgr.m_cellCol.compareTo(end) > 0) {
  return (false);
```

```
  }
  if (mgr.m_cellRow.compareTo("V") > 0) {
    return (false);
  }

  int m = m_rowLetters.indexOf(mgr.m_cellRow);
  int n = sequence.indexOf(mgr.m_cellCol);

  double gridNorthing = 100000.0 * m + falseNorthing;
  double gridEasting = 100000.0 * (n + 1);
  //
  if (gridNorthing >= 2000000.0) {
    gridNorthing = gridNorthing - 2000000.0;
  }
  LatitudeBand band = getBandByLetter(mgr.m_row);
  double scaledMinNorthing = band.m_minNorthing;
  while (scaledMinNorthing >= 2000000.0) {
    scaledMinNorthing = scaledMinNorthing - 2000000.0;
  }
  gridNorthing = gridNorthing - scaledMinNorthing;
  if (gridNorthing < 0.0) {
    gridNorthing = gridNorthing + 2000000.0;
  }
  gridNorthing = band.m_minNorthing + gridNorthing;
  utm.m_easting = gridEasting + mgr.m_easting;
  utm.m_northing = gridNorthing + mgr.m_northing;
  DecimalFormat form = new DecimalFormat("00");
  utm.m_utmZone = form.format(mgr.m_zone) + mgr.m_row;
  //
  return (true);
}

public static void main(String[] args) {
  UTM utm = new UTM();
  String sequence = "ABCDEFGH";
  String rowLetters = "ABCDEFGHJKLMNPQRSTUV";
  System.out.println("-------------------------------------------");
  for (int j = 0; j < rowLetters.length(); j++) {
    for (int i = 0; i < sequence.length(); i++) {
      String mgr = "01N" + sequence.substring(i, i + 1) +
          rowLetters.substring(j, j + 1) + "1200015000";
      MRGtoUTM(mgr, utm);
      System.out.println("MRG=" + mgr);
      System.out.println("UTM=" + utm);
      Position p = utm.toPosition();
```

```
        System.out.println(" LL=" + p);
        System.out.println("-----------------------------------------------");
      }
    }
  }
}

package com.sun.BaseSigactEJB;

import java.text.*;
import java.util.*;

public class Position {

  static private DecimalFormat form1 = new DecimalFormat("000");
  static private DecimalFormat form2 = new DecimalFormat("00");
  static private DecimalFormat form3 = new DecimalFormat("00.0");

  private double m_lat;
  private double m_lng;

  public Position() {
    m_lat = 0;
    m_lng = 0;
  }

  public Position(double lat, double lng) {
    m_lat = lat;
    m_lng = lng;
  }

  /**
   * Set the Latitude and Longitude of a position.  These valuse are in the
   * range -90 to 90 degrees and -180 to 180 degress, respectively.
   * @param lat Latitude between -90 and 90 degrees.
   * @param lng Longitude between -180 and 180 degrees.
   */
  public void setLatLng(double lat, double lng) {
    m_lat = lat;
    m_lng = lng;
  }

  /**
   * Get the Latitude of the position.
   * @return Latitude double.
```

```
 */
public double getLat() {
  return (m_lat);
}

/**
 * Get the Longitude of the position.
 * @return Longitude double.
 */
public double getLng() {
  return (m_lng);
}

public String toString() {
  return (getString("DD:MM:SS.SN DDD:MM:SS.SW"));
}

public String getString(String format) {
  String latString;
  String lngString;
  if (StringOps.patternMatch("DDMMN DDDMMW", format)) {
    {
      double lat = m_lat;
      double sgn = 1.0;
      if (lat < 0.0) {
        sgn = -1;
        lat = -lat;
      }
      int totalMinutes = (int) (60 * lat + .5);
      int totalDegrees = totalMinutes / 60;
      int minutes = totalMinutes - 60 * totalDegrees;
      String deg = form2.format(totalDegrees);
      String min = form2.format(minutes);
      if (sgn == 1.0) {
        latString = deg + min + "N";
      }
      else {
        latString = deg + min + "S";
      }
    }
    {
      double lng = m_lng;
      double sgn = 1.0;
      if (lng < 0.0) {
        sgn = -1;
```

```
      lng = -lng;
    }
    int totalMinutes = (int) (60 * lng + .5);
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form1.format(totalDegrees);
    String min = form2.format(minutes);
    if (sgn == 1.0) {
      lngString = deg + min + "E";
    }
    else {
      lngString = deg + min + "W";
    }
  }
  return (latString + " " + lngString);
}
if (StringOps.patternMatch("DDMMSSN DDDMMSSW", format)) {
  {
    double lat = m_lat;
    double sgn = 1.0;
    if (lat < 0.0) {
      sgn = -1;
      lat = -lat;
    }
    int totalSeconds = (int) (3600 * lat + .5);
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form2.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form2.format(seconds);
    if (sgn == 1.0) {
      latString = deg + min + sec + "N";
    }
    else {
      latString = deg + min + sec + "S";
    }
  }
  {
    double lng = m_lng;
    double sgn = 1.0;
    if (lng < 0.0) {
      sgn = -1;
      lng = -lng;
```

```
    }
    int totalSeconds = (int) (3600 * lng + .5);
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form1.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form2.format(seconds);
    if (sgn == 1.0) {
      lngString = deg + min + sec + "E";
    }
    else {
      lngString = deg + min + sec + "W";
    }
  }
  return (latString + " " + lngString);
}
if (StringOps.patternMatch("DD:MM:SSN DDD:MM:SSW", format)) {
  {
    double lat = m_lat;
    double sgn = 1.0;
    if (lat < 0.0) {
      sgn = -1;
      lat = -lat;
    }
    int totalSeconds = (int) (3600 * lat + .5);
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form2.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form2.format(seconds);
    if (sgn == 1.0) {
      latString = deg + ":" + min + ":" + sec + "N";
    }
    else {
      latString = deg + ":" + min + ":" + sec + "S";
    }
  }
  {
    double lng = m_lng;
    double sgn = 1.0;
    if (lng < 0.0) {
```

```
      sgn = -1;
      lng = -lng;
    }
    int totalSeconds = (int) (3600 * lng + .5);
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form1.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form2.format(seconds);
    if (sgn == 1.0) {
      lngString = deg + ":" + min + ":" + sec + "E";
    }
    else {
      lngString = deg + ":" + min + ":" + sec + "W";
    }
  }
  return (latString + " " + lngString);
}
if (StringOps.patternMatch("DDMMSS.SN DDDMMSS.SW", format)) {
  {
    double lat = m_lat;
    double sgn = 1.0;
    if (lat < 0.0) {
      sgn = -1;
      lat = -lat;
    }
    int totalTenthsOfSeconds = (int) Math.round(36000.0 * lat);
    int tenths = totalTenthsOfSeconds % 10;
    int totalSeconds = totalTenthsOfSeconds / 10;
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form2.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form3.format(seconds + (double) tenths / 10.0);
    if (sgn == 1.0) {
      latString = deg + min + sec + "N";
    }
    else {
      latString = deg + min + sec + "S";
    }
  }
```

```
  {
    double lng = m_lng;
    double sgn = 1.0;
    if (lng < 0.0) {
      sgn = -1;
      lng = -lng;
    }
    int totalTenthsOfSeconds = (int) Math.round(36000.0 * lng);
    int tenths = totalTenthsOfSeconds % 10;
    int totalSeconds = totalTenthsOfSeconds / 10;
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form1.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form3.format(seconds + (double) tenths / 10.0);
    if (sgn == 1.0) {
      lngString = deg + min + sec + "E";
    }
    else {
      lngString = deg + min + sec + "W";
    }
  }
  return (latString + " " + lngString);
}
if (StringOps.patternMatch("DD:MM:SS.SN DDD:MM:SS.SW", format)) {
  {
    double lat = m_lat;
    double sgn = 1.0;
    if (lat < 0.0) {
      sgn = -1;
      lat = -lat;
    }
    int totalTenthsOfSeconds = (int) Math.round(36000.0 * lat);
    int tenths = totalTenthsOfSeconds % 10;
    int totalSeconds = totalTenthsOfSeconds / 10;
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form2.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form3.format(seconds + (double) tenths / 10.0);
    if (sgn == 1.0) {
```

```
      latString = deg + ":" + min + ":" + sec + "N";
     }
     else {
      latString = deg + ":" + min + ":" + sec + "S";
     }
    }
    {
     double lng = m_lng;
     double sgn = 1.0;
     if (lng < 0.0) {
      sgn = -1;
      lng = -lng;
     }
     int totalTenthsOfSeconds = (int) Math.round(36000.0 * lng);
     int tenths = totalTenthsOfSeconds % 10;
     int totalSeconds = totalTenthsOfSeconds / 10;
     int totalMinutes = totalSeconds / 60;
     int seconds = totalSeconds - 60 * totalMinutes;
     int totalDegrees = totalMinutes / 60;
     int minutes = totalMinutes - 60 * totalDegrees;
     String deg = form1.format(totalDegrees);
     String min = form2.format(minutes);
     String sec = form3.format(seconds + (double) tenths / 10.0);
     if (sgn == 1.0) {
      lngString = deg + ":" + min + ":" + sec + "E";
     }
     else {
      lngString = deg + ":" + min + ":" + sec + "W";
     }
    }
    return (latString + " " + lngString);
  }
  return ("");
}

static public String getLatString(double lat, String format) {
  String latString;
  if (StringOps.patternMatch("DDMMSS.SN", format)) {
    {
     double sgn = 1.0;
     if (lat < 0.0) {
      sgn = -1;
      lat = -lat;
     }
     int totalTenthsOfSeconds = (int) Math.round(36000.0 * lat);
```

```
    int tenths = totalTenthsOfSeconds % 10;
    int totalSeconds = totalTenthsOfSeconds / 10;
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form2.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form3.format(seconds + (double) tenths / 10.0);
    if (sgn == 1.0) {
      latString = deg + min + sec + "N";
    }
    else {
      latString = deg + min + sec + "S";
    }
  }
  return (latString);
}
if (StringOps.patternMatch("DDMMSSN", format)) {
  {
    double sgn = 1.0;
    if (lat < 0.0) {
      sgn = -1;
      lat = -lat;
    }
    int totalSeconds = (int) (3600 * lat + .5);
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form2.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form2.format(seconds);
    if (sgn == 1.0) {
      latString = deg + min + sec + "N";
    }
    else {
      latString = deg + min + sec + "S";
    }
  }
  return (latString);
}
if (StringOps.patternMatch("DDMMN", format)) {
  {
    double sgn = 1.0;
```

```
    if (lat < 0.0) {
      sgn = -1;
      lat = -lat;
    }
    int totalMinutes = (int) (60 * lat + .5);
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form2.format(totalDegrees);
    String min = form2.format(minutes);
    if (sgn == 1.0) {
      latString = deg + min + "N";
    }
    else {
      latString = deg + min + "S";
    }
  }
  return (latString);
}
if (StringOps.patternMatch("DD:MM:SS.SN", format)) {
  {
    double sgn = 1.0;
    if (lat < 0.0) {
      sgn = -1;
      lat = -lat;
    }
    int totalTenthsOfSeconds = (int) Math.round(36000.0 * lat);
    int tenths = totalTenthsOfSeconds % 10;
    int totalSeconds = totalTenthsOfSeconds / 10;
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form2.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form3.format(seconds + (double) tenths / 10.0);
    if (sgn == 1.0) {
      latString = deg + ":" + min + ":" + sec + "N";
    }
    else {
      latString = deg + ":" + min + ":" + sec + "S";
    }
  }
  return (latString);
}
if (StringOps.patternMatch("DD:MM:SSN", format)) {
```

```
    {
     double sgn = 1.0;
     if (lat < 0.0) {
       sgn = -1;
       lat = -lat;
     }
     int totalSeconds = (int) (3600 * lat + .5);
     int totalMinutes = totalSeconds / 60;
     int seconds = totalSeconds - 60 * totalMinutes;
     int totalDegrees = totalMinutes / 60;
     int minutes = totalMinutes - 60 * totalDegrees;
     String deg = form2.format(totalDegrees);
     String min = form2.format(minutes);
     String sec = form2.format(seconds);
     if (sgn == 1.0) {
       latString = deg + ":" + min + ":" + sec + "N";
     }
     else {
       latString = deg + ":" + min + ":" + sec + "S";
     }
    }
    return (latString);
  }
  return ("");
}

static public String getLngString(double lng, String format) {
  String lngString;
  if (StringOps.patternMatch("DDDMMSS.SW", format)) {
    {
     double sgn = 1.0;
     if (lng < 0.0) {
       sgn = -1;
       lng = -lng;
     }
     int totalTenthsOfSeconds = (int) Math.round(36000.0 * lng);
     int tenths = totalTenthsOfSeconds % 10;
     int totalSeconds = totalTenthsOfSeconds / 10;
     int totalMinutes = totalSeconds / 60;
     int seconds = totalSeconds - 60 * totalMinutes;
     int totalDegrees = totalMinutes / 60;
     int minutes = totalMinutes - 60 * totalDegrees;
     String deg = form1.format(totalDegrees);
     String min = form2.format(minutes);
     String sec = form3.format(seconds + (double) tenths / 10.0);
```

```
    if (sgn == 1.0) {
      lngString = deg + min + sec + "E";
    }
    else {
      lngString = deg + min + sec + "W";
    }
  }
  return (lngString);
}
if (StringOps.patternMatch("DDDMMSSW", format)) {
  {
    double sgn = 1.0;
    if (lng < 0.0) {
      sgn = -1;
      lng = -lng;
    }
    int totalSeconds = (int) (3600 * lng + .5);
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form1.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form2.format(seconds);
    if (sgn == 1.0) {
      lngString = deg + min + sec + "E";
    }
    else {
      lngString = deg + min + sec + "W";
    }
  }
  return (lngString);
}
if (StringOps.patternMatch("DDDMMW", format)) {
  {
    double sgn = 1.0;
    if (lng < 0.0) {
      sgn = -1;
      lng = -lng;
    }
    int totalMinutes = (int) (60 * lng + .5);
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form1.format(totalDegrees);
    String min = form2.format(minutes);
```

```
    if (sgn == 1.0) {
      lngString = deg + min + "E";
    }
    else {
      lngString = deg + min + "W";
    }
  }
  return (lngString);
}
if (StringOps.patternMatch("DDD:MM:SS.SW", format)) {
  {
    double sgn = 1.0;
    if (lng < 0.0) {
      sgn = -1;
      lng = -lng;
    }
    int totalTenthsOfSeconds = (int) Math.round(36000.0 * lng);
    int tenths = totalTenthsOfSeconds % 10;
    int totalSeconds = totalTenthsOfSeconds / 10;
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
    int totalDegrees = totalMinutes / 60;
    int minutes = totalMinutes - 60 * totalDegrees;
    String deg = form1.format(totalDegrees);
    String min = form2.format(minutes);
    String sec = form3.format(seconds + (double) tenths / 10.0);
    if (sgn == 1.0) {
      lngString = deg + ":" + min + ":" + sec + "E";
    }
    else {
      lngString = deg + ":" + min + ":" + sec + "W";
    }
  }
  return (lngString);
}
if (StringOps.patternMatch("DDD:MM:SSW", format)) {
  {
    double sgn = 1.0;
    if (lng < 0.0) {
      sgn = -1;
      lng = -lng;
    }
    int totalSeconds = (int) (3600 * lng + .5);
    int totalMinutes = totalSeconds / 60;
    int seconds = totalSeconds - 60 * totalMinutes;
```

```
      int totalDegrees = totalMinutes / 60;
      int minutes = totalMinutes - 60 * totalDegrees;
      String deg = form1.format(totalDegrees);
      String min = form2.format(minutes);
      String sec = form2.format(seconds);
      if (sgn == 1.0) {
        lngString = deg + ":" + min + ":" + sec + "E";
      }
      else {
        lngString = deg + ":" + min + ":" + sec + "W";
      }
    }
    return (lngString);
  }

  return ("");
}

public static Random m_random;
static {
  m_random = new Random(System.currentTimeMillis());
}

static public Position getRandomPosition(double lat, double lng,
                                         double radius) {
  lat += radius * (m_random.nextDouble() - 0.5);
  lng += radius * (m_random.nextDouble() - 0.5);
  return (new Position(lat, lng));
}

public static Position StringToPosition(String pos) {
  Location location = new Location(pos);
  Position position = new Position();
  if (location.getPosition(position)) {
    return (position);
  }
  return (null);
}

public static void main(String[] args) {
  //m_lng: double = -134.87933333333334
  {
    Position p = Location.StringToPosition("422200.0S 1345245.6W");
    p.m_lng = -134.87933333333334;
    System.out.println(p.getString("DDMMSS.SN DDDMMSS.SW"));
```

```
      p.m_lng = -134.8793333333333;
      System.out.println(p.getString("DDMMSS.SN DDDMMSS.SW"));
    }
    {
      Position p = Location.StringToPosition("UT:01NCD1200015000");
      System.out.println(p.getString("DDMMSS.SN DDDMMSS.SW"));
    }
  }

}

package com.sun.BaseSigactEJB;


import java.io.*;

public class StringOps {

  static double DegreesToRadians = Math.PI / 180.0;

  static public final String validDigit = "0123456789";
  static public final String validAlpha = "ABCDEFGHJKLMNPQRSTUVWXYZ";

//         static public final String validChars =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ.-
#*%,;:&@`'[]\\^,:$=!<>()+?\"{}~_|";
  static public final String validChars =
      "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ.- #*%,;:";

  static public boolean isValidOTH(String str) {
    for (int i = 0; i < str.length(); i++) {
      String s = str.substring(i, i + 1);
      if (validChars.indexOf(s) == -1) {
        return (false);
      }
    }
    return (true);
  }

  //
  //
==========================================================
======
```

```
  //
================================================================
======
  //
  static public boolean isValidOctal(String val, int nchar) {
    if (val.length() > nchar) {
      return (false);
    }
    try {
      int n = Integer.parseInt(val, 8);
      return (true);
    }
    catch (Exception e) {
      return (false);
    }
  }

  //
  //
================================================================
======
  //
================================================================
======
  //
  static public String toValidOTH(String str, int maxChar) {
    String result = "";
    str = str.toUpperCase().trim();
    for (int i = 0; i < str.length(); i++) {
      String s = str.substring(i, i + 1);
      if (validChars.indexOf(s) == -1) {
        if (!result.endsWith(" ")) {
          result += " ";
        }
      }
      else {
        if (! (result.endsWith(" ") && s.equals(" "))) {
          result += s;
        }
      }
    }
    // Remove all double spaces
    str = result;
    result = "";
    for (int i = 0; i < str.length(); i++) {
```

```
    String s = str.substring(i, i + 1);
    if (result.endsWith(" ") && s.equals(" ")) {
      continue;
    }
    result += s;
  }
  if (result.length() > maxChar) {
    result = result.substring(0, maxChar);
  }
  return (result.trim());
}

static public boolean patternMatch(String pattern, String target) {
  int n = pattern.length();
  int m = target.length();
  if (m < n) {
    return (false);
  }

  for (int i = 0; i < n; i++) {
    String p = pattern.substring(i, i + 1);
    String q = target.substring(i, i + 1);
    if (p.equals("*")) {
      continue;
    }
    else if (p.equals("A")) {
      if (validAlpha.indexOf(q) == -1) {
        return (false);
      }
    }
    else if (p.equals("#")) {
      if (validDigit.indexOf(q) == -1) {
        return (false);
      }
    }
    else if (p.equals("N") || p.equals("S")) {
      if (!q.equals("N") && !q.equals("S")) {
        return (false);
      }
    }
    else if (p.equals("E") || p.equals("W")) {
      if (!q.equals("E") && !q.equals("W")) {
        return (false);
      }
    }
```

```
    else {
      if (!p.equals(q)) {
        return (false);
      }
    }
  }
  return (true);
}

static public void writeString(java.io.ObjectOutput out, String str) throws
    IOException {
  out.write(str.length());
  out.writeBytes(str);
}

static public String readString(java.io.ObjectInput in) throws IOException,
    ClassNotFoundException {
  int length = in.read();
  byte data[] = new byte[length];
  in.read(data);
  return (new String(data));
}

static public String padString(String in, int nchars) {
  String str = in;
  while (nchars > str.length()) {
    str = str + " ";
  }
  return (str.substring(0, nchars));
}

static public String padString(String in, String pattern, int nchars) {
  String str = in;
  while (nchars > str.length()) {
    str = str + pattern;
  }
  return (str.substring(0, nchars));
}

static public String truncate(String in, int nchars) {
  if (in.length() <= nchars) {
    return (in);
  }
  return (in.substring(0, nchars));
}
```

```
  static public int findArg(int start, String args[], String name) {
    for (int i = start; i < args.length; i++) {
      if (args[i].equalsIgnoreCase(name)) {
        return (i);
      }
    }
    return ( -1);
  }

  //
  //
=================================================================
======
  //
=================================================================
======
  //
  static public boolean isValidLetterString(String str) {
    for (int i = 0; i < str.length(); i++) {
      if (!Character.isLetter(str.charAt(i))) {
        return (false);
      }
    }
    return (true);
  }

  //
  //
=================================================================
======
  //
=================================================================
======
  //
  static public boolean isValidLetterOrDigitString(String str) {
    for (int i = 0; i < str.length(); i++) {
      if (!Character.isLetterOrDigit(str.charAt(i))) {
        return (false);
      }
    }
    return (true);
  }

  //
```

```
  //
================================================================
======
  //
================================================================
======
  //
  static public boolean isValidDigitString(String str) {
    for (int i = 0; i < str.length(); i++) {
      if (!Character.isDigit(str.charAt(i))) {
        return (false);
      }
    }
    return (true);
  }

  //
  //
================================================================
======
  //
================================================================
======
  //
  public static String concat(String strings[], String delim) {
    if (strings.length == 0) {
      return ("");
    }
    String result = strings[0];
    for (int i = 1; i < strings.length; i++) {
      result += (delim + strings[i]);
    }
    return (result);
  }

  //
  //
================================================================
======
  //
================================================================
======
  //
  public static String concat(String strings[]) {
    return (concat(strings, ","));
```

```
  }

  //
  //
//================================================================
======
  //
//================================================================
======
  //
  public static String concat(String one, String two) {
    if (one == null) {
      return (two);
    }
    return (one + "," + two);
  }

  //
  //
//================================================================
======
  //
//================================================================
======
  //
  public static String concat(String one, String two, String delim) {
    if (one == null) {
      return (two);
    }
    return (one + delim + two);
  }

  //
  //
//================================================================
======
  //
//================================================================
======
  //
  static public String convertToValidName(String instanceID) {
    instanceID = instanceID.replaceAll("\\.", "_");
    instanceID = instanceID.replaceAll("-", "_");
    return (instanceID.toUpperCase());
  }
```

```
  //
  //
=============================================================
=========
  //
=============================================================
=========
  //
  static public String instanceIDtoDomain(String instanceID) {
    String parts[] = instanceID.split("\\.");
    return (parts[0]);
  }

  //
  //
=============================================================
======
  //
=============================================================
======
  //
  public static void main(String[] args) {
    String result = StringOps.toValidOTH("CLASS    NAME", 11);
    System.out.println(result);
  }
}
package com.sun.BaseSigactEJB;

import java.text.*;



public class UTM {
  //
  static final double PI = 3.14159265;
  static final double FOURTHPI = PI / 4.0;
  static final double deg2rad = PI / 180.0;
  static final double rad2deg = 180.0 / PI;

  //
  public double m_northing = 0;
  public double m_easting = 0;
  public String m_utmZone = "31N";
```

```
//
public UTM() {}

//
public String toString() {
  DecimalFormat format = new DecimalFormat("00000000.000");
  String result = m_utmZone;
  result = result + " " + format.format(m_northing);
  result = result + " " + format.format(m_easting);
  return (result);
}

//
static public boolean LLtoUTM(int ReferenceEllipsoid, double Lat, double Long,
                              UTM utm) {
  //
  // converts lat/long to UTM coords.  Equations from USGS Bulletin 1532
  // Written by Chuck Gantz- chuck.gantz@globalstar.com
  //
  char zone = UTMLetterDesignator(Lat);
  if (zone == 'Z') {
    return (false);
  }
  if (ReferenceEllipsoid == 0 ||
      ReferenceEllipsoid >= Ellipsoid.m_ElipseTable.length) {
    return (false);
  }
  //
  double a = Ellipsoid.m_ElipseTable[ReferenceEllipsoid].m_EquatorialRadius;
  double eccSquared = Ellipsoid.m_ElipseTable[ReferenceEllipsoid].
      m_eccentricitySquared;
  double k0 = 0.9996;
  double LongOrigin;
  double eccPrimeSquared;
  double N, T, C, A, M;
  //Make sure the longitude is between -180.00 .. 179.9
  while (Long < -180.0) {
    Long += 360.0;
  }
  while (Long >= 180.0) {
    Long -= 360.0;
    //
  }
  double LatRad = Lat * deg2rad;
  double LongRad = Long * deg2rad;
```

```
    double LongOriginRad;
    int ZoneNumber;

    ZoneNumber = (int) ( (Long + 180.0) / 6.0) + 1;

    if (Lat >= 56.0 && Lat < 64.0 && Long >= 3.0 && Long < 12.0) {
      ZoneNumber = 32;

      // Special zones for Svalbard
    }
    if (Lat >= 72.0 && Lat < 84.0) {
      if (Long >= 0.0 && Long < 9.0) {
        ZoneNumber = 31;
      }
      else if (Long >= 9.0 && Long < 21.0) {
        ZoneNumber = 33;
      }
      else if (Long >= 21.0 && Long < 33.0) {
        ZoneNumber = 35;
      }
      else if (Long >= 33.0 && Long < 42.0) {
        ZoneNumber = 37;
      }
    }
    LongOrigin = (ZoneNumber - 1.0) * 6.0 - 180.0 + 3.0; // +3 puts origin in middle of
zone
    LongOriginRad = LongOrigin * deg2rad;

    // compute the UTM Zone from the latitude and longitude
    utm.m_utmZone = "" + ZoneNumber + zone;

    eccPrimeSquared = (eccSquared) / (1 - eccSquared);
    double sinLat = Math.sin(LatRad);
    double sin2Lat = Math.sin(2 * LatRad);
    double sin4Lat = Math.sin(4 * LatRad);
    double sin6Lat = Math.sin(6 * LatRad);
    double cosLat = Math.cos(LatRad);
    double tanLat = Math.tan(LatRad);
    N = a / Math.sqrt(1 - eccSquared * sinLat * sinLat);
    T = tanLat * tanLat;
    C = eccPrimeSquared * cosLat * cosLat;
    A = cosLat * (LongRad - LongOriginRad);

    double E2 = eccSquared * eccSquared;
    double E3 = eccSquared * E2;
```

```
  M = a *
     ( (1.0 - eccSquared / 4.0 - 3.0 * E2 / 64.0 - 5.0 * E3 / 256.0) * LatRad
      -
      (3.0 * eccSquared / 8.0 + 3.0 * E2 / 32.0 + 45.0 * E3 / 1024.0) * sin2Lat
      + (15.0 * E2 / 256.0 + 45.0 * E3 / 1024.0) * sin4Lat
      - (35.0 * E3 / 3072.0) * sin6Lat);
  //
  double A2 = A * A;
  double A3 = A * A2;
  double T2 = T * T;
  utm.m_easting = (k0 * N *
                (A + (1.0 - T + C) * A3 / 6.0 +
                 (5.0 - 18 * T + T2 + 72 * C - 58 * eccPrimeSquared) * A2 *
                 A3 / 120.0) + 500000.0);
  utm.m_northing = (k0 *
                 (M +
                  N * tanLat *
                  (A2 / 2 + (5 - T + 9 * C + 4 * C * C) * A2 * A2 / 24.0 +
                   (61.0 - 58.0 * T + T2 + 600 * C -
                    330.0 * eccPrimeSquared) * A3 * A3 / 720.0)));
  if (Lat < 0) {
    utm.m_northing += 10000000.0; //10000000 meter offset for southern hemisphere
  }
  return (true);
}

static char UTMLetterDesignator(double Lat) {
  //
  // This routine determines the correct UTM letter designator for the given latitude
  // returns 'Z' if latitude is outside the UTM limits of 84N to 80S
  // Written by Chuck Gantz- chuck.gantz@globalstar.com
  //
  char LetterDesignator;
  //
  if ( (84 >= Lat) && (Lat >= 72)) {
    LetterDesignator = 'X';
  }
  else if ( (72 > Lat) && (Lat >= 64)) {
    LetterDesignator = 'W';
  }
  else if ( (64 > Lat) && (Lat >= 56)) {
    LetterDesignator = 'V';
  }
  else if ( (56 > Lat) && (Lat >= 48)) {
    LetterDesignator = 'U';
```

```
}
else if ( (48 > Lat) && (Lat >= 40)) {
  LetterDesignator = 'T';
}
else if ( (40 > Lat) && (Lat >= 32)) {
  LetterDesignator = 'S';
}
else if ( (32 > Lat) && (Lat >= 24)) {
  LetterDesignator = 'R';
}
else if ( (24 > Lat) && (Lat >= 16)) {
  LetterDesignator = 'Q';
}
else if ( (16 > Lat) && (Lat >= 8)) {
  LetterDesignator = 'P';
}
else if ( (8 > Lat) && (Lat >= 0)) {
  LetterDesignator = 'N';
}
else if ( (0 > Lat) && (Lat >= -8)) {
  LetterDesignator = 'M';
}
else if ( ( -8 > Lat) && (Lat >= -16)) {
  LetterDesignator = 'L';
}
else if ( ( -16 > Lat) && (Lat >= -24)) {
  LetterDesignator = 'K';
}
else if ( ( -24 > Lat) && (Lat >= -32)) {
  LetterDesignator = 'J';
}
else if ( ( -32 > Lat) && (Lat >= -40)) {
  LetterDesignator = 'H';
}
else if ( ( -40 > Lat) && (Lat >= -48)) {
  LetterDesignator = 'G';
}
else if ( ( -48 > Lat) && (Lat >= -56)) {
  LetterDesignator = 'F';
}
else if ( ( -56 > Lat) && (Lat >= -64)) {
  LetterDesignator = 'E';
}
else if ( ( -64 > Lat) && (Lat >= -72)) {
  LetterDesignator = 'D';
```

```
    }
    else if ( ( -72 > Lat) && (Lat >= -80)) {
      LetterDesignator = 'C';
    }
    else {
      LetterDesignator = 'Z'; // This is here as an error flag to show that the Latitude is
outside the UTM limits
       //
    }
    return LetterDesignator;
  }

  //
  public Position toPosition() {
    Position p = new Position();
    if (UTMtoLL(23, this, p)) {
      return (p);
    }
    return (p);
  }

  //
  public static boolean UTMtoLL(UTM utm, Position p) {
    return (UTMtoLL(23, utm, p));
  }

  //
  public static boolean UTMtoLL(int ReferenceEllipsoid, UTM utm, Position p) {
    //
    // converts UTM coords to lat/long.  Equations from USGS Bulletin 1532
    // Written by Chuck Gantz- chuck.gantz@globalstar.com
    //
    double Lat = 0.0;
    double Long = 0.0;
    double k0 = 0.9996;
    double a = Ellipsoid.m_ElipseTable[ReferenceEllipsoid].m_EquatorialRadius;
    double eccSquared = Ellipsoid.m_ElipseTable[ReferenceEllipsoid].
        m_eccentricitySquared;
    double eccPrimeSquared;
    double root = Math.sqrt(1.0 - eccSquared);
    double e1 = (1.0 - root) / (1 + root);
    double N1, T1, C1, R1, D, M;
    double LongOrigin;
    double mu, phi1, phi1Rad;
    double x, y;
```

```
   int ZoneNumber;
   char ZoneLetter;
   boolean NorthernHemisphere;

   x = utm.m_easting - 500000.0; // remove 500,000 meter offset for longitude
   y = utm.m_northing;
   int n = utm.m_utmZone.length();
   if (! (n == 2 || n == 3)) {
     return (false);
   }

   ZoneNumber = Integer.parseInt(utm.m_utmZone.substring(0, n - 1));
   ZoneLetter = utm.m_utmZone.charAt(n - 1);
   if ( (ZoneLetter - 'N') >= 0) {
     NorthernHemisphere = true;
   }
   else {
     NorthernHemisphere = false;
     y -= 10000000.0; // remove 10,000,000 meter offset used for southern hemisphere
   }

   LongOrigin = (ZoneNumber - 1.0) * 6.0 - 180.0 + 3.0; //+3 puts origin in middle of
zone

   eccPrimeSquared = (eccSquared) / (1 - eccSquared);

   double E2 = eccSquared * eccSquared;
   double E3 = eccSquared * E2;
   M = y / k0;
   mu = M / (a * (1.0 - eccSquared / 4.0 - 3.0 * E2 / 64.0 - 5.0 * E3 / 256.0));
   //
   double sin2mu = Math.sin(2 * mu);
   double sin4mu = Math.sin(4 * mu);
   double sin6mu = Math.sin(6 * mu);
   double Q2 = e1 * e1;
   double Q3 = e1 * Q2;
   phi1Rad = mu + (3.0 * e1 / 2.0 - 27.0 * Q3 / 32.0) * sin2mu
      + (21.0 * Q2 / 16.0 - 55.0 * Q2 * Q2 / 32.0) * sin4mu
      + (151.0 * Q3 / 96.0) * sin6mu;
   phi1 = phi1Rad * rad2deg;
   //
   double sinphi = Math.sin(phi1Rad);
   double sinphi2 = sinphi * sinphi;
   double cosphi = Math.cos(phi1Rad);
   double cosphi2 = cosphi * cosphi;
```

```
    double tanphi = Math.tan(phi1Rad);
    double tanphi2 = tanphi * tanphi;
    N1 = a / Math.sqrt(1.0 - eccSquared * sinphi2);
    T1 = tanphi2;
    C1 = eccPrimeSquared * cosphi2;
    R1 = a * (1.0 - eccSquared) / Math.pow(1.0 - eccSquared * sinphi2, 1.5);
    D = x / (N1 * k0);
    //
    double C2 = C1 * C1;
    double D2 = D * D;
    double D3 = D * D2;
    //
    Lat = phi1Rad -
       (N1 * tanphi / R1) *
       (D2 / 2.0 -
        (5.0 + 3.0 * T1 + 10.0 * C1 - 4.0 * C2 - 9.0 * eccPrimeSquared) * D2 *
        D2 / 24.0
        +
        (61.0 + 90.0 * T1 + 298.0 * C1 + 45.0 * T1 * T1 - 252.0 * eccPrimeSquared -
         3.0 * C2) * D3 * D3 / 720.0);
    Lat = Lat * rad2deg;
    //
    Long = (D - (1.0 + 2.0 * T1 + C1) * D3 / 6.0 +
           (5.0 - 2.0 * C1 + 28.0 * T1 - 3.0 * C2 + 8.0 * eccPrimeSquared +
            24.0 * T1 * T1) * D2 * D3 / 120.0) / cosphi;
    Long = LongOrigin + Long * rad2deg;
    p.setLatLng(Lat, Long);
    //
    return (true);
  }

  //
  public static void main(String[] args) {
    UTM utm = new UTM();
//                  double lat = 41+0/60.0+29/3600.0;
//                  double lng = 34+33/60.0+31/3600.0;
    // UTM = 36T 04540821.210 00631067.899
//                  double lat = 41+54/60.0+10/3600.0;
//                  double lng = 115+15/60.0+50/3600.0;
    // UTM = 50T 04640439.156 00355998.335
    double lat = -21.5794444;
    double lng = 27.6691667;
    // UTM = 35K 07613572.678 00569274.910
    System.out.println(lat + " " + lng);
    if (UTM.LLtoUTM(23, lat, lng, utm)) {
```

```
      System.out.println("UTM = " + utm);
      Position p = new Position();
      if (UTM.UTMtoLL(23, utm, p)) {
        System.out.println(p.getLat() + " " + p.getLng());
      }
    }
    else {
      System.out.println("Invalid UTM");
    }
  }

}
/*
 * sigact.java
 *
 * Created on May 6, 2007, 1:04 PM
 * LCDR Kurt Rothenhaus
 * Dissertation Source Code
 *
 */

package com.sun.BaseSigactEJB;

/**
 *
 * @author kjrothen
 */
public class sigact {

    /** Creates a new instance of sigact */
    public sigact() {}


    private String AttackTarget;
    private double SigLat;
    private double SigLon;

    public String getAttackTarget(){
        return AttackTarget;
    }

     public void setAttackTarget(String AttackTarget){
        this.AttackTarget = AttackTarget;
    }
```

```
    public double getSigLat(){
       return SigLat;
    }
    public void setSigLat(double SigLat){
       this.SigLat = SigLat;
    }

     public double getSigLon(){
       return SigLon;
    }
    public void setSigLon(double SigLon){
       this.SigLon = SigLon;
    }
}
    /*
 * FusionEngineGeo.java
 * Kurt Rothenhaus
 *  * Created on May 6, 2007, 11:25 AM
 * Used in conjuntion with research on AutoData Fusion in a SOA
 * This class takes input from two data sources and approximates the position
 */
package com.sun.GeoTrackFusion;


import java.util.ArrayList;
import java.util.Iterator;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebParam.Mode;
import javax.jws.WebService;
import java.sql.Timestamp;
import java.util.List;
import javax.xml.ws.Holder;
/**
 *
 * @author kjrothen
 */

@Stateless()
@WebService()
public class GeoTrackFusion {
```

```
    /**
     * Web service operation
     */
@WebMethod
public List<combined> isGeoTrackFusion(List<sigact> sigbatch, List<project>
projbatch){


        //Return array of new tracks

        List<combined> fusebatch = new ArrayList();
        fusebatch = getfusedBatch(sigbatch,projbatch);
return fusebatch;


}



public List<combined> getfusedBatch(List<sigact> sigbatch, List<project> projbatch) {

        List<combined> comie = new ArrayList<combined>();

        int iterSig =0;
        double siglat = 0;
        double siglon = 0;
        double projlat = 0;
        double projlon = 0;
        String newname;
        boolean maketrack = false;

        Iterator<sigact> sigIter = sigbatch.iterator();
        while(sigIter.hasNext()) {
            sigact s = sigIter.next();
            siglat = s.getSigLat();
            siglon = s.getSigLon();
            Iterator<project> projIter = projbatch.iterator();
            while(projIter.hasNext()) {
                project p = projIter.next();
                projlat = p.getProjLat();
                projlon = p.getProjLon();
                // maketrack fourth varialble is a distance setting in Miles.
                maketrack = isnextTo(projlat,projlon,siglat,siglon,1000);
                if (maketrack){
                    newname = p.getProjectName() + s.getAttackTarget();
```

```
                double newlat = (projlat + siglat)/2;
                double newlon = (projlon + siglon)/2;

                  combined com = new combined();
                  com.setcombName(newname);
                  com.setcombLat(newlat);
                  com.setcombLon(newlon);
                  comie.add(com);

                }// END IF
            }
        }


        return comie;


    }

public boolean isnextTo(double projlat, double projlon, double lat, double lon, double
spininitial) {

            //add spinner value to projlat/projlon to get area
          // String spin1 = jSpinner2.getValue().toString();

           double spin = spininitial * .01;

           double top_left_lat =  projlat +
spin;//Double.parseDouble(ratMapperFrame.jTextField5.getText());
           double top_left_lon =  projlon -
spin;//Double.parseDouble(ratMapperFrame.jTextField1.getText());
           double Bot_right_lat = projlat -
spin;//Double.parseDouble(ratMapperFrame.jTextField4.getText());
           double Bot_right_lon = projlon +
spin;//Double.parseDouble(ratMapperFrame.jTextField2.getText());

           System.out.println("project =" + projlat + "/" +projlon+"@"+ spin);
           System.out.println("**********************************************");

            System.out.println("top_left_lat =" + top_left_lat);
            System.out.println("top_left_lon =" + top_left_lon);
            System.out.println("Bot_right_lat =" + Bot_right_lat);
            System.out.println("Bot_right_lon =" + Bot_right_lon);
```

```
        if ((lat >= Bot_right_lat) && (lat <= top_left_lat)
                && (lon <= Bot_right_lon) && (lon >= top_left_lon)){

    System.out.println("True");
          return true;

                }
       else {
                System.out.println("false");
                return false;
       }
   }



}
/*
 * sigact.java
 *
 * Created on May 6, 2007, 1:04 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package com.sun.GeoTrackFusion;

/**
 *
 * @author kjrothen
 */
public class combined {

   /** Creates a new instance of sigact */
   public combined() {}


   private String combName;
   private double combLat;
   private double combLon;

   public String getcombName(){
      return combName;
   }
```

```
    public void setcombName(String combName){
      this.combName = combName;
    }

    public double getcombLat(){
      return combLat;
    }
    public void setcombLat(double combLat){
      this.combLat = combLat;
    }

    public double getcombLon(){
      return combLon;
    }
    public void setcombLon(double combLon){
      this.combLon = combLon;
    }
}

/*
 * project.java
 *
 * Created on May 6, 2007, 1:04 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package com.sun.GeoTrackFusion;
/**
 *
 * @author kjrothen
 */
public class project {

    /** Creates a new instance of sigact */
    public project() {}


    private String ProjectName;
    private double ProjLat;
    private double ProjLon;

    public String getProjectName(){
      return ProjectName;
```

```
    }

     public void setProjectName(String ProjectName){
        this.ProjectName = ProjectName;
    }

    public double getProjLat(){
        return ProjLat;
    }
    public void setProjLat(double ProjLat){
        this.ProjLat = ProjLat;
    }

     public double getProjLon(){
        return ProjLon;
    }
    public void setProjLon(double ProjLon){
        this.ProjLon = ProjLon;
    }
}
/*
 * NewTrackService.java
 *
 * Created on April 10, 2007, 12:15 AM
 * LCDR Kurt Rothenhaus
 * Created in support of Dissertation Research
 * and open the template in the editor.
 * Accepts new track objects and establishes persistence via a JDBC
 * connection.
 */

package com.sun.ReconProjEJB;
import java.util.List;
import java.util.ArrayList;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebParam.Mode;
import javax.jws.WebService;
import javax.xml.ws.Holder;
import com.borland.dx.sql.dataset.*;
import com.borland.dbswing.*;

@WebService
@Stateless
```

```
public class ReconProjectEJB{


    Database database1 = new Database();
    QueryDataSet queryDataSet1 = new QueryDataSet();
    DBDisposeMonitor dBDisposeMonitor1 = new DBDisposeMonitor();



@WebMethod
public List<RetVal> getReconProject(@WebParam(name="Sector") String SECTOR){

       List<RetVal> projbatch = new ArrayList();
       return projbatch = getProjBatch(SECTOR);
   }

  public List<RetVal> getProjBatch(String SECTOR){

      queryDataSet1.close();

   queryDataSet1.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(database1,
"SELECT
RMS_ALL.SHORT_NAME,RMS_ALL.URI,RMS_ALL.GRID_LOCATION" +
   " FROM \"C:\\Documents and Settings\\kjrothen\\My
Documents\\RMS_ALL\".RMS_ALL"
    + " WHERE SECTOR = '"+SECTOR+ "' ", null, true, Load.ALL));

  database1.setConnection(new
com.borland.dx.sql.dataset.ConnectionDescriptor("jdbc:odbc:RMS_ALL", "", "", false,
"sun.jdbc.odbc.JdbcOdbcDriver"));
  database1.setUseCaseSensitiveId(true);
  database1.setUseSpacePadding(false);
  database1.setDatabaseName("");
    queryDataSet1.open();
   String MGRS = "null";


      List<RetVal> progie = new ArrayList<RetVal>();

      while (queryDataSet1.inBounds()) {

        MGRS = queryDataSet1.getString("GRID_LOCATION");
        String projectName = queryDataSet1.getString("URI");
        Position R = MGRStoLat(MGRS);
          double userlat = R.getLat();
```

```
          double userlon = R.getLng();



      RetVal project = new RetVal();
        project.setURI(projectName);
        project.setHVULat(userlat);
        project.setHVULong(userlon);
        progie.add(project);
       queryDataSet1.next();
       }

       queryDataSet1.close();
   return progie;
   }


public Position MGRStoLat(String Mgrs){


    UTM utm = new UTM();

   try {
    MGR.MRGtoUTM(Mgrs, utm);
    System.out.println("MRG=" + Mgrs);
    System.out.println("UTM=" + utm);
    Position p = utm.toPosition();
    System.out.println(" LL=" + p);
    double lat = p.getLat();
    double lng = p.getLng();
    return p;
   }
    catch (Exception e) {
      MGR.MRGtoUTM("01NGD1200015000", utm);
      Position p = utm.toPosition();
      System.out.println("Catch the exception =" + p);
      return p;
      }
    }



}
```

```
   /*
 * ReconProjecEJB.java
 *
 * Created on April 10, 2007, 12:15 AM
 * LCDR Kurt Rothenhaus
 * Created in support of Disertation Research
 * and open the template in the editor.
 */

package com.sun.NewTrackService;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
@Stateless
public class NewTrackService{




@WebMethod
public String newTrackdb(List<combined> fusebatch){
   Connection conn = null;
   double comlat = 0.0;
   double comlon = 0.0;
   String comname = null;


try{
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
} catch(java.lang.ClassNotFoundException e) {
   System.err.print("ClassNotFoundException: ");
   System.err.println(e.getMessage());
}

try{
   conn = DriverManager.getConnection("jdbc:odbc:COMBINED");
}catch(SQLException ex){
   System.err.println("database connection: " + ex.getMessage());
}


    Iterator<combined> comIter = fusebatch.iterator();
       while(comIter.hasNext()) {
          combined c = comIter.next();
          comlat = c.getcombLat();
          comlon = c.getcombLon();
          comname = c.getcombName();

   Statement stmt;
         try {
            stmt = conn.createStatement();
        stmt.executeUpdate("INSERT INTO Combined VALUES
("'+comname+"',"'+comlat+"',"'+comlon+"')");
         } catch (SQLException ex) {
            ex.printStackTrace();
         }

       }


   return "New Tracks added to Database ";
     }
}
```

```
/*
 * ReconProjecEJB.java
 *
 * Created on April 10, 2007, 12:15 AM
 * LCDR Kurt Rothenhaus
```

```
 * Created in support of Disertation Research
 * and open the template in the editor.
 */

package com.sun.ReconProjEJB;
import java.util.List;
import java.util.ArrayList;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebParam.Mode;
import javax.jws.WebService;
import javax.xml.ws.Holder;
import com.borland.dx.sql.dataset.*;
import com.borland.dbswing.*;

@WebService
@Stateless
public class ReconProjectEJB{


    Database database1 = new Database();
    QueryDataSet queryDataSet1 = new QueryDataSet();
    DBDisposeMonitor dBDisposeMonitor1 = new DBDisposeMonitor();



@WebMethod
public List<RetVal> getReconProject(@WebParam(name="Sector") String SECTOR){

       List<RetVal> projbatch = new ArrayList();
       return projbatch = getProjBatch(SECTOR);
   }

  public List<RetVal> getProjBatch(String SECTOR){

      queryDataSet1.close();

   queryDataSet1.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(database1,
"SELECT
RMS_ALL.SHORT_NAME,RMS_ALL.URI,RMS_ALL.GRID_LOCATION" +
   " FROM \"C:\\Documents and Settings\\kjrothen\\My
Documents\\RMS_ALL\".RMS_ALL"
    + " WHERE SECTOR = '"+SECTOR+ "' ", null, true, Load.ALL));
```

```
  database1.setConnection(new
com.borland.dx.sql.dataset.ConnectionDescriptor("jdbc:odbc:RMS_ALL", "", "", false,
"sun.jdbc.odbc.JdbcOdbcDriver"));
  database1.setUseCaseSensitiveId(true);
  database1.setUseSpacePadding(false);
  database1.setDatabaseName("");
    queryDataSet1.open();
   String MGRS = "null";


       List<RetVal> progie = new ArrayList<RetVal>();

       while (queryDataSet1.inBounds()) {

         MGRS = queryDataSet1.getString("GRID_LOCATION");
         String projectName = queryDataSet1.getString("URI");
         Position R = MGRStoLat(MGRS);
           double userlat = R.getLat();
           double userlon = R.getLng();



      RetVal project = new RetVal();
         project.setURI(projectName);
         project.setHVULat(userlat);
         project.setHVULong(userlon);
         progie.add(project);
       queryDataSet1.next();
       }

       queryDataSet1.close();
   return progie;
   }


public Position MGRStoLat(String Mgrs){


    UTM utm = new UTM();

   try {
    MGR.MRGtoUTM(Mgrs, utm);
    System.out.println("MRG=" + Mgrs);
    System.out.println("UTM=" + utm);
    Position p = utm.toPosition();
```

```
   System.out.println(" LL=" + p);
   double lat = p.getLat();
   double lng = p.getLng();
   return p;
  }

   catch (Exception e) {
     MGR.MRGtoUTM("01NGD1200015000", utm);
     Position p = utm.toPosition();
     System.out.println("Catch the exception =" + p);
     return p;
     }
   }



}
```

**********BPEL**********

```
<?xml version="1.0" encoding="UTF-8"?>
<process
   name="GeoTrackFusion"
   targetNamespace="http://enterprise.netbeans.org/bpel/GeoTrackFusion"
   xmlns="http://schemas.xmlsoap.org/ws/2004/03/business-process/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:bpws="http://schemas.xmlsoap.org/ws/2004/03/business-process/"
   xmlns:wsdlNS="http://enterprise.netbeans.org/bpel/GeoTrackFusion"
xmlns:ns1="http://j2ee.netbeans.org/wsdl/GeoTrackFusion"
xmlns:ns2="http://BaseSigactEJB.sun.com/" xmlns:ns3="http://ReconProjEJB.sun.com/"
xmlns:ns4="http://GeoTrackFusion.sun.com/"
xmlns:ns5="http://NewTrackService.sun.com/"
xmlns:ns0="http://xml.netbeans.org/schema/newXMLSchema">
   <import namespace="http://j2ee.netbeans.org/wsdl/GeoTrackFusion"
location="GeoTrackFusion.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
   <import namespace="http://BaseSigactEJB.sun.com/"
location="Partners/BaseSigactEJB/BaseSigactEJB.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
   <import namespace="http://ReconProjEJB.sun.com/"
location="Partners/ReconProjectEJB/ReconProjectEJB.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
   <import namespace="http://GeoTrackFusion.sun.com/"
location="Partners/GeoTrackFusion/GeoTrackFusion.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
```

```
   <import namespace="http://NewTrackService.sun.com/"
location="Partners/NewTrackService/NewTrackService.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
   <partnerLinks>
      <partnerLink name="NewTrackEJB"
partnerLinkType="ns5:NewTrackServiceLinkType"
partnerRole="NewTrackServiceRole"/>
      <partnerLink name="GeoFuserEJB"
partnerLinkType="ns4:GeoTrackFusionLinkType"
partnerRole="GeoTrackFusionRole"/>
      <partnerLink name="ReconEJB"
partnerLinkType="ns3:ReconProjectEJBLinkType"
partnerRole="ReconProjectEJBRole"/>
      <partnerLink name="SigactEJB" partnerLinkType="ns2:BaseSigactEJBLinkType"
partnerRole="BaseSigactEJBRole"/>
      <partnerLink name="GeoTrackFusionBpel"
partnerLinkType="ns1:GeoTrackFusionPartner"
myRole="GeoTrackFusionPortTypeRole"/>
   </partnerLinks>
   <variables>
      <variable name="GeoTrackFusionOperationOutputfin"
messageType="ns1:GeoTrackFusionOperationReply"/>
      <variable name="NewTrackdbOutput" messageType="ns5:newTrackdbResponse"/>
      <variable name="NewTrackdbInput" messageType="ns5:newTrackdb"/>
      <variable name="IsGeoTrackFusionOutput"
messageType="ns4:isGeoTrackFusionResponse"/>
      <variable name="IsGeoTrackFusionInput"
messageType="ns4:isGeoTrackFusion"/>
      <variable name="GetReconProjectOutput"
messageType="ns3:getReconProjectResponse"/>
      <variable name="GetReconProjectInput" messageType="ns3:getReconProject"/>
      <variable name="ProcessBaseSIGACTEJBOutput"
messageType="ns2:processBaseSIGACTEJBResponse"/>
      <variable name="ProcessBaseSIGACTEJBInput"
messageType="ns2:processBaseSIGACTEJB"/>
      <variable name="GeoTrackFusionOperationInput"
messageType="ns1:GeoTrackFusionOperationRequest"/>
   </variables>
   <sequence>
      <receive name="ReceiveFromBpel" createInstance="yes"
partnerLink="GeoTrackFusionBpel" operation="GeoTrackFusionOperation"
portType="ns1:GeoTrackFusionPortType" variable="GeoTrackFusionOperationInput"/>
      <assign name="Assign1">
         <copy>
```

```
<from>$GeoTrackFusionOperationInput.Fusionmessage/ns0:SigactStart</from>
            <to>$ProcessBaseSIGACTEJBInput.parameters/arg0</to>
          </copy>
          <copy>

<from>$GeoTrackFusionOperationInput.Fusionmessage/ns0:SigactFinish</from>
            <to>$ProcessBaseSIGACTEJBInput.parameters/arg1</to>
          </copy>
          <copy>
            <from>$GeoTrackFusionOperationInput.Fusionmessage/ns0:Sector</from>
            <to>$GetReconProjectInput.parameters/Sector</to>
          </copy>
        </assign>
        <flow name="Flow1">
          <invoke name="InvokeSigactEJB" partnerLink="SigactEJB"
operation="processBaseSIGACTEJB" portType="ns2:BaseSigactEJB"
inputVariable="ProcessBaseSIGACTEJBInput"
outputVariable="ProcessBaseSIGACTEJBOutput"/>
          <invoke name="InvokeReconEJB" partnerLink="ReconEJB"
operation="getReconProject" portType="ns3:ReconProjectEJB"
inputVariable="GetReconProjectInput" outputVariable="GetReconProjectOutput"/>
        </flow>
        <assign name="Assign2">
          <copy>

<from>$ProcessBaseSIGACTEJBOutput.parameters/return/attackTarget</from>
            <to>$IsGeoTrackFusionInput.parameters/arg0/attackTarget</to>
          </copy>
          <copy>
            <from>$ProcessBaseSIGACTEJBOutput.parameters/return/sigLat</from>
            <to>$IsGeoTrackFusionInput.parameters/arg0/sigLat</to>
          </copy>
          <copy>
            <from>$ProcessBaseSIGACTEJBOutput.parameters/return/sigLon</from>
            <to>$IsGeoTrackFusionInput.parameters/arg0/sigLon</to>
          </copy>
          <copy>
            <from>$GetReconProjectOutput.parameters/return/URI</from>
            <to>$IsGeoTrackFusionInput.parameters/arg1/projectName</to>
          </copy>
          <copy>
            <from>$GetReconProjectOutput.parameters/return/HVULong</from>
            <to>$IsGeoTrackFusionInput.parameters/arg1/projLon</to>
          </copy>
```

```
        <copy>
          <from>$GetReconProjectOutput.parameters/return/HVULat</from>
          <to>$IsGeoTrackFusionInput.parameters/arg1/projLat</to>
        </copy>
      </assign>
      <invoke name="InvokeGeoFuserEJB" partnerLink="GeoFuserEJB"
operation="isGeoTrackFusion" portType="ns4:GeoTrackFusion"
inputVariable="IsGeoTrackFusionInput" outputVariable="IsGeoTrackFusionOutput"/>
      <assign name="Assign5">
        <copy>
          <from>$IsGeoTrackFusionOutput.parameters/return/combLat</from>
          <to>$NewTrackdbInput.parameters/arg0/combLat</to>
        </copy>
        <copy>
          <from>$IsGeoTrackFusionOutput.parameters/return/combLon</from>
          <to>$NewTrackdbInput.parameters/arg0/combLon</to>
        </copy>
        <copy>
          <from>$IsGeoTrackFusionOutput.parameters/return/combName</from>
          <to>$NewTrackdbInput.parameters/arg0/combName</to>
        </copy>
      </assign>
      <invoke name="InvokeNewTrackEJB" partnerLink="NewTrackEJB"
operation="newTrackdb" portType="ns5:NewTrackService"
inputVariable="NewTrackdbInput" outputVariable="NewTrackdbOutput"/>
      <assign name="Assign6">
        <copy>
          <from>$NewTrackdbOutput.parameters/return</from>
          <to>$GeoTrackFusionOperationOutputfin.FusionResponse/ns0:return</to>
        </copy>
      </assign>
      <reply name="ReplytoBpel" partnerLink="GeoTrackFusionBpel"
operation="GeoTrackFusionOperation" portType="ns1:GeoTrackFusionPortType"
variable="GeoTrackFusionOperationOutputfin"/>
  </sequence>

</process>

**********GeoTrackFusion wsdl**********
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://j2ee.netbeans.org/wsdl/GeoTrackFusion"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://j2ee.netbeans.org/wsdl/GeoTrackFusion" name="GeoTrackFusion"
xmlns:ns="http://xml.netbeans.org/schema/newXMLSchema"
xmlns:plink="http://schemas.xmlsoap.org/ws/2004/03/partner-link/">
   <types>
      <xsd:schema targetNamespace="http://j2ee.netbeans.org/wsdl/GeoTrackFusion">
         <xsd:import namespace="http://xml.netbeans.org/schema/newXMLSchema"
schemaLocation="GeoTrackFusion.xsd"/>
      </xsd:schema>
   </types>
   <message name="GeoTrackFusionOperationRequest">
      <part name="Fusionmessage" element="ns:processGeoTrackFusion"/>
   </message>
   <message name="GeoTrackFusionOperationReply">
      <part name="FusionResponse" element="ns:processGeoTrackFusionResponse"/>
   </message>
   <portType name="GeoTrackFusionPortType">
      <operation name="GeoTrackFusionOperation">
         <input name="input1" message="tns:GeoTrackFusionOperationRequest"/>
         <output name="output1" message="tns:GeoTrackFusionOperationReply"/>
      </operation>
   </portType>
   <binding name="GeoTrackFusionBinding" type="tns:GeoTrackFusionPortType">
      <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GeoTrackFusionOperation">
         <soap:operation/>
         <input name="input1">
            <soap:body use="literal"/>
         </input>
         <output name="output1">
            <soap:body use="literal"/>
         </output>
      </operation>
   </binding>
   <service name="GeoTrackFusionService">
      <port name="GeoTrackFusionPort" binding="tns:GeoTrackFusionBinding">
         <soap:address
location="http://localhost:18181/GeoTrackFusionService/GeoTrackFusionPort"/>
      </port>
   </service>
   <plink:partnerLinkType name="GeoTrackFusionPartner">
      <!-- partnerLinkType are automatically generated when a new portType is added.
partnerLinkType are used by BPEL processes.
```

In a BPEL process, a partner link represents the interaction between the BPEL process and a partner service. Each partner link is associated with a partner link type.
A partner link type characterizes the conversational relationship between two services. The partner link type can have one or two roles.-->

```
      <plink:role name="GeoTrackFusionPortTypeRole"
portType="tns:GeoTrackFusionPortType"/>
   </plink:partnerLinkType>
</definitions>
```

**********GeoTrackFusion.xsd**********

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://xml.netbeans.org/schema/newXMLSchema"
          xmlns:tns="http://xml.netbeans.org/schema/newXMLSchema"
          elementFormDefault="qualified">
   <xsd:complexType name="processGeoTrackFusion">
      <xsd:sequence>
         <xsd:element name="SigactStart" nillable="true" type="xsd:string"/>
         <xsd:element name="SigactFinish" nillable="true" type="xsd:string"/>
         <xsd:element name="Sector" nillable="true" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="processGeoTrackFusionResponse">
      <xsd:sequence>
         <xsd:element name="return" nillable="true" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="processGeoTrackFusionResponse"
type="tns:processGeoTrackFusionResponse"/>
   <xsd:element name="processGeoTrackFusion" type="tns:processGeoTrackFusion"/>
</xsd:schema>
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Alberts, D, Garstka, J, Stein, F (1999). *Net-Centric Capabilities*. Washington, DC: CCRP Publication Series.

Alberts, D. & Hayes R. (2003). *Power to the Edge*. Washington, DC: CCRP Publication Series.

Barry, D. (2004). *Service-oriented architecture (SOA) definition*. Retrieved May 9, 2008, from http://www.service-architecture.com

Barry, D. (2004). *Web Services definition*. Retrieved May 9, 2008,

Bennett, T. (2004). *Pentagons New Map*. New York, NY: Putnam's Sons.

Bequet, H. (2001). *Professional Java Soap.* Chicago: Peer Information Inc.

Bowman, C. & Steinberg, A. (2001) *A Systems Engineering approach for implementing Data Fusion Systems.* In D. Hall and J. Llinas, *Handbook of Multi-sensor Data Fusion* (16-1 to 16-38). Boca Raton, FL: CRC Press.

Buschmann, F., Henney, K., Schmidt, D. (2007). *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*, Hoboken, NJ: John Wiley & Sons Ltd.

Czarnecki, K. & Eisenecker, U. (2000). *Generative Programming, Methods, Tools, and Applications*, Boston, MA: Addison-Wesley.

Department of Defense. (2003). *Net-Centric Data Strategy*, Washington, DC: U.S. Government Printing Office. http://www.defenselink.mil/cio-nii/docs/Net-Centric-Data-Strategy-2003-05-092.pdf, Retrieved January 10, 2008

Department of Homeland Defense. (2007) *National Concept of Operations for Maritime Domain Awareness*, Washington, DC: U.S. Government Printing Office. http://www.dhs.gov/xlibrary/assets/HSPD_MDAPlan.pdf, Retrieved January 10, 2008

Department of the Navy Chief Information Officer. (2005). *Department Of The Navy Information Technology Applications And Data Management* (SECNAVINST 5000.36A). Washington, DC: U.S. Government Printing Office.

Department of the Navy. (1996). *Understanding Link-16, A guidebook for New Users*. Washington, DC: U.S. Government Printing Office

Engel, D., Rothenhaus, K., Settelmayer, G. (2006). *eXtensible Common Operations Picture*, Space and Naval Warfare System Center Biannual Review.

Erl, T. (2005). *Service Oriented Architecture Concepts, Technology, and Design*. Saddle River, NJ: Prentice Hall.
from http://www.service-architecture.com/web services/articles/web_services_definition.html

Gaines, L. & Michael, J. (2005, November). Service Level Agreements as Vehicles for Managing Acquisition of Software-Intensive Systems, *Defense Acquisition Journal*, 285-305.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design Patterns-Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley.

Graham, S., David, D. (2005). *Building Web Services with Java*, London: Sams Publishing.

Hobbes, R.(2004) *Internet Timeline*, 2004, Retrieved May 9, 2008, from http://www.zakon.org/robert/internet/timeline/

Hobbins, T. (2007, August). Cultural shift [Electronic Version]. *C4ISR Journal*, 117-123.

Humphrey, S., & Monteiro, M. (2002). *Rumble in the jungle: J2EE versus .Net, Part 1, How do J2EE and Microsoft's .Net compare in enterprise environments.* Retrieved May 9, 2008, http://www.javaworld.com/javaworld/jw-06-2002/jw-0628-j2eevnet.html

Monson-Haefel, R., & Chappell, D. (2001). *JAVA Message Service.* New York, NY: O'Reilly.

Moore, P. (2002). *Characteristics of a Service Oriented Architecture*. New York, NY: Irx Limited.

Office of Naval Research (1991). *Functional Description of the Data Fusion Process*. Washington, DC: U.S. Government Printing Office.

Pirsig, R., (1991*). Lila: An Inquiry into Morals*, New York, NY: Bantam Books.

Pressman, S. (2001). *Software Engineering, A Practitioner's Approach Fifth Edition*. New York, NY: McGraw Hill.

Steinberg, A., Bowman, C. (2001). *Revisions to the JDL Data Fusion Mode,* In D. Hall and J. Llinas, *Handbook of Multi-sensor Data Fusion* (pp. 2-1-2-19). Boca Raton, FL: CRC Press.

U.S. Department of Defense. (2004*). Department of Defense Architecture Framework Deskbook*. Washington, DC: U.S. Government Printing Office.

United States Coast Guard (2002*). Applications For Equipment Authorization Of Universal Shipborne Automatic Identification Systems To be Coordinated with U.S. Coast Guard To Ensure Homeland Security*. Washington, DC: U.S. Government Printing Office. Retrieved August 8, 2007, from http://www.navcen.uscg.gov/marcomms/ais.htm.

Willard, R. (2002, October). The Art of Command and Control. *Proceedings*, 128, 68-73.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor James Bret Michael
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

4. Professor Man-Tak Shing
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

5. Professor Ted Lewis
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

6. Professor John Osmundson
   Department of Information Sciences
   Naval Postgraduate School

7. Dr. David Engel
   Code 273
   SPAWAR System Center San Diego

8. Dr. Frank White
   MITRE Corporation

9. John Shea
   Technical Director
   Navy Program Office for ISR (PEO C4I/PMW-120)

10. CAPT D.J. Legoff
    Deputy Program Manager
    Navy Program Office for Command and Control (PEO C4I/PMW-150)

11. CAPT Dubois, USN (ret)
NGA/JPSIO

12. LTC Carl Oros, USMC
Marine Coprs Representative
Department of Information Sciences
Naval Postgraduate School